Efficacité énergétique et ordonnancement des systèmes temps-réel multiprocesseurs.

K. Bhatti

UNIVERSITÉ DE NICE - SOPHIA ANTIPOLIS

École Doctorale STIC

Sciences et Technologies de l'Information et de la Communication

THÈSE

pour obtenir le titre de

Docteur en Sciences

de l'Université de Nice - Sophia Antipolis

Mention Informatique

présentée et soutenue par

Muhammad Khurram BHATTI

Energy-aware Scheduling for Multiprocessor Real-time Systems

Thèse dirigée par Cécile BELLEUDY

Laboratoire LEAT, Université de Nice-Sophia Antipolis -CNRS, Sophia Antipolis

soutenue le 18 avril 2011, devant le jury composé de:

Président du Jury	Lionel Torres	Pr. Université de Montpellier-II, France
Rapporteurs	Isabelle Puaut	Pr. Université de Rennes-I, France
	Guy Gogniat	Pr. Université de Bretagne Sud, France
Examinateurs	Yvon Trinquet	Pr. Université de Nantes, France
	Lionel Torres	Pr. Université de Montpellier-II, France
	Michel Auguin	DR. CNRS, France (Co-directeur de thèse)
Directeur de thèse	Cécile Belleudy	Maître de Conférences, Université de Nice-Sophia Antipolis, France

Abstract

Real-time applications have become more sophisticated and complex in their behavior and interaction over the time. Contemporaneously, multiprocessor architectures have emerged to handle these sophisticated applications. Inevitably, these complex real-time systems, encompassing a range from small-scale embedded devices to large-scale data centers, are increasingly challenged to reduce energy consumption while maintaining assurance that timing constraints will be met. To address this issue in real-time systems, many software-based approaches such as dynamic voltage and frequency scaling and dynamic power management have emerged. Yet their flexibility is often matched by the complexity of the solution, with the accompanying risk that deadlines will occasionally be missed. As the computational demands of real-time embedded systems continue to grow, effective yet transparent energy-management approaches will become increasingly important to minimize energy consumption, extend battery life, and reduce thermal losses. We believe that power- and energy-efficiency and scheduling of real-time systems are closely related problems, which should be tackled together for best results. By exploiting the characteristic parameters of real-time application tasks, the energy-consciousness of scheduling algorithms and the quality of service of real-time applications can be significantly improved.

To support our thesis, this dissertation proposes novel approaches for energy-management within the paradigm of energy-aware scheduling for soft and hard real-time applications, which are scheduled over identical multiprocessor platforms. Our first contribution is a *Two-level Hierarchical Scheduling Algorithm (2L-HiSA)* for multiprocessor systems, which falls in the category of restricted-migration scheduling. 2L-HiSA addresses the sub-optimality of EDF scheduling algorithm in multiprocessors by dividing the problem into a two-level hierarchy of schedulers. Our second contribution is a dynamic power management technique, called the *Assertive Dynamic Power Management (AsDPM)* technique. AsDPM serves as an admission control technique for real-time tasks, which decides when exactly a ready task shall execute, thereby reducing the number of active processors, which eventually reduces energy consumption. Our third contribution is a dynamic voltage and frequency scaling technique, called the *Deterministic Stretch-to-Fit (DSF)* technique, which falls in the category of inter-task DVFS techniques and works in conjunction with global scheduling algorithms. DSF comprises an online *Dynamic Slack Reclamation algorithm (DSR)*, an *Online Speculative speed adjustment Mechanism (OSM)*, and an *m-Task Extension (m-TE)* technique. Our fourth and final contribution is a generic power/energy management scheme for multiprocessor systems, called the *Hybrid Power Management (HyPowMan)* scheme. HyPowMan serves as a top-level entity that, instead of designing new power/energy management policies (whether DPM or DVFS) for specific operating conditions, takes a set of well-known existing policies. Each policy in the selected policy set performs well for a given set of operating conditions. At runtime, the best-performing policy for given workload is adapted by the HyPowMan scheme through a machine-learning algorithm.

To my father Ismail, who always dared to dream,
and to my mother Mumtaz (late) for making it a reality.

Acknowledgments

Completion of my PhD required countless selfless acts of support, generosity, and time by people in my personal and academic life. I can only attempt to humbly acknowledge and thank the people and institutions that have given so freely throughout my PhD career and made this dissertation possible. I am thankful to the Higher Education Commission (HEC) of Pakistan for providing uninterrupted funding support throughout my Masters and PhD career. I am sincerely thankful to Cécile Belleudy, my advisor, for being a constant source of invaluable encouragement, aid, and expertise during my years at University of Nice. While many students are fortunate to find a single mentor, I have been blessed with two. I am deeply grateful to Michel Auguin, my co-advisor, for the guidance, support, respect, and kindness that he has shown me over the last four years. The mentoring, friendship, and collegiality of both Cécile and Michel enriched my academic life and have left a profound impression on how academic research and collaboration should ideally be conducted.

I am extremely thankful to the members of my dissertation committee. Guy Gogniat and Isabelle Puaut have graciously accepted to serve on the committee as reviewers and provided unique feedback, comments, and questions on multiprocessor scheduling, along with a lot of encouragement. Yvon Trinquet has provided me with wise advice and support throughout my PhD and also accepted to be a part of my dissertation committee. I have always admired his precise questions and unique manner of addressing difficult research problems. Lionel Torres has been very kind for accepting to be the president of dissertation committee and a source of insightful comments and ideas to my research and its effective presentation. I must acknowledge that all these people have greatly inspired me. Other colleagues who I owe gratitude for their support of my research or major PhD milestones include: Sébastian Bilavarn, Francois Verdier, Ons Mbarek, and Jabran Khan. I am also grateful to the always helpful LEAT research laboratory and University of Nice staff. I would like to thank all my research collaborators who have enhanced my enthusiasm and understanding of real-time systems through various projects, namely; the collaborators of Pherma, COMCAS, and STORM tool design and development projects. Since the path through the PhD program would be much more difficult without examples of success, I am indebted to Muhammad Farooq who has given friendship and guidance as recent real-time system PhD graduate from LEAT.

My family and friends have been an unending source of love and inspiration throughout my PhD career. My father, Ismail, has offered unconditional understanding and encouragement. My sisters, Shaista and Sofia, have kept me sane with their humor and understanding even from distance. My brother, Asad, has been a great and selfless support to me throughout these year of my absence from home. My friends in French Riviera, Najam, Naveed, Uzair, Sabir, Chafic, Umer, Siouar, Khawla, Amel, Alice, and Sébastian have provided hours of enjoyable distraction from my work. I will always remember the time I have shared with them. Lastly, I can only wish if my mother, Mumtaz, was still alive to embrace me on achieving this milestone. She will always remain my constant.

Contents

List of Algorithms

List of Figures

List of Tables

Symbols and Acronyms

Symbols	Definition
t	Time instant
τ	Task set
T_i	Individual task indexed as i
$T_{i,j}$	Individual job j of task T_i
J	Job set
r_i	Release time of task T_i
C_i	Worst-case execution time (WCET) of task T_i
d_i	Relative deadline of task T_i
P_i	Period of task T_i
O_i	Offset of first job $T_{i,1}$ of T_i w.r.t. system activation
L_i	Absolute laxity of task T_i
l_i	Anticipative laxity of task T_i
u_i	Utilization of individual task T_i
$U_{sum}(\tau)$	Utilization of task set τ
π_k	Individual processor indexed as k
Π	Processor set/ platform
n	Number of tasks in τ
m	Number of processors in Π
ν	Speed of processor π_k
F_{op}	Operating frequency
V_{op}	Operating voltage
V_{th}	Threshold voltage
E	Energy
ε	Dynamic slack
ϕ	Scaling factor
$Pwr(\nu)$	Power as function of speed ν
τ_{π_k}	Subset of tasks partitioned on processor π_k
S^{can}	Canonical Schedule of tasks
S^{pra}	Practical Schedule of tasks
$DBF(\tau, L)$	Demand Bound Function of task set τ over interval of length L
N	Number of Experts (where, expert is any power management scheme)
w	Weight factor for individual expert
W	Weight vector for expert set

continued on next page

– *continued from previous page*

Acronyms	Description
h	Probability factor for individual expert
H	Probability vector for expert set
AET	Actual Execution Time
AsDPM	Assertive Dynamic Power Management
ASEDZL	Anticipating Slack Earliest Deadline until Zero Laxity
BCET	Best-case Execution Time
BET	Break-Even Time
ccEDF	Cycle-conserving Earliest Deadline First
DeTQ	Deferred Tasks Queue
DPM	Dynamic Power Management
DSF	Deterministic Stretch-to-Fit
DSR	Dynamic Slack Reclamation
DVFS	Dynamic Voltage and Frequency Scaling
EDF	Earliest Deadline First
EDZL	Earliest Deadline until Zero Laxity
HyPowMan	Hybrid Power Management
LLF	Least Laxity First
LLREF	Least Local Remaining Execution First
m-TE	m-Task Extension
OSM	Online Speculative speed adjustment Mechanism
PFair	Proportionate Fairness
ReTQ	Ready Tasks Queue
RM	Rate Monotonic
RuTQ	Running Tasks Queue
SMP	Symmetric shared-memory Multiprocessor
TQ	Tasks Queue
WCET	Worst-case Execution Time
2L-HiSA	Two-level Hierarchical Scheduling Algorithm

Part I

Complete dissertation: English version

Chapter 1

Introduction

Contents

1.1 Introduction

In real-time systems, the temporal correctness of produced output is equally important as the logical correctness [42]. That is, real-time systems must not only perform correct operations, but also perform them at correct time. A logically correct operation performed by a system can result in either an erroneous, completely useless, or degraded output depending upon the strictness of time constraints. Based on the level of strictness of timing constraints, real-time systems can be classified into three broad categories: *hard* real-time, *soft* real-time, and *firm* real-time systems[47, 77, 105]. Such systems must be predictable and provably temporally correct. The designer must verify that the system is correct prior to runtime –i.e., for instance, for any possible execution of a hard real-time system, each execution results in all deadlines being met. Even for the simplest systems, the number of possible execution scenarios is either infinite or prohibitively large. Therefore, exhaustive simulation or testing cannot be used to verify the temporal correctness of such systems. Instead, formal analysis techniques are necessary to ensure that the designed systems are, by construction, provably temporally correct and predictable [42, 47]. Over the time, real-time applications have become more sophisticated and complex in their behavior and interaction. Contemporaneously, multi-core architectures have emerged to handle these sophisticated applications and since then, prevailed in many commercial systems. Although significant research has been focused on the design of real-time systems during past decades, the emergence of multi-core architectures have renewed some existing challenges as well as brought some new ones for real-time research community. These challenges can be classified into three broad categories: multiprocessor platform architecture design, multiprocessor scheduling, and multiprocessor energy-efficiency.

As the multiprocessor architectures are already widely used, it becomes more and more clear that future real-time systems will be deployed on multiprocessor

architectures. Multiprocessor architectures have certain new features that must be taken into consideration. For instance, application programs executing on different cores usually share fine-grained resources, like shared caches, interconnect networks, and shared memory bandwidth, making the conventional design practices not suitable to multi-core systems. Thus, multi-core architectures are significantly challenging in their design, analysis, and implementation.

Another challenge for real-time systems is the scheduling problem. The real-time scheduling problem on multiprocessor models is very different from and significantly more difficult than single-processor scheduling. Single-processor scheduling algorithms cannot be applied on multiprocessor systems without loss of *optimality*. A scheduling algorithm is said to be *optimal* if it can successfully schedule any *feasible* task system [105]. A task system is said to be feasible if it is guaranteed that a schedule exists that meets all deadlines of all jobs, for all sequences of jobs that can be generated by the task system. Optimality of scheduling algorithms is a critical design issue in multiprocessor real-time systems as under-utilized platform resources are not desirable. Multiprocessor scheduling algorithms employ either a *partitioned* or *global* scheduling approach (or hybrids of the two). Partitioned scheduling, under which tasks are statically assigned to processors and scheduled on each processor using single-processor scheduling algorithms, have low scheduling overheads. However, the management of globally-shared resources such as a shared main memory and caches can become quite difficult under partitioning, precisely because each processor is scheduled independently. Moreover, partitioning tasks to processors is equivalent to solving a bin-packing problem: on an m-processor system, each task with a size equal to its utilization must be placed into one of m bins of size one representing a processor. Bin-packing is considered a strong NP-hard problem [60]. In global scheduling algorithms, on the other hand, all processors select jobs to schedule from a single run queue. As a result, jobs may migrate among processors, and contention for shared data structures is likely. Until recently, no multiprocessor optimal global scheduling algorithm existed before the proposition of PFair and its heuristic algorithms in [13, 106]. Although few recently proposed algorithms are known to be optimal [13, 106, 77, 28], multiprocessor scheduling theory has many fundamental problems still open to address.

The ever-increasing complexity of real-time applications that are being scheduled over multiprocessor architectures, ranging from multimedia and telecommunication to aerospace applications, poses another great challenge –i.e., the power consumption rate of computing devices which has been increasing exponentially. Power densities in microprocessors have almost doubled every three years [103, 56]. This increased power usage poses two types of difficulties: the energy consumption and rise in device's temperature. As energy is power integrated over time, supplying the required energy may become prohibitively expensive, or even technologically infeasible. This is a particular difficulty in portable systems that heavily rely on batteries for energy, and will become even more critical as battery capacities are

increasing at a much slower rate than power consumption. The energy consumed in computing devices is in large part converted into heat. With processing platforms heading towards 3D-stacked architectures [30, 104], thermal imbalances and energy consumption in modern chips have resulted in power becoming a first-class design constraint for modern embedded real-time systems. Therefore, complex real-time systems must reduce energy consumption while providing guarantees that the timing constraints will be met. Energy management in real-time systems has been addressed from both hardware and software points of view. Many software-based approaches, particularly scheduling-based approaches such as Dynamic Voltage and Frequency Scaling (DVFS) and Dynamic Power Management (DPM) have been proposed by real-time research community over the past few years. Yet their flexibility is often matched by the complexity of the solution, with the accompanying risk that deadlines will occasionally be missed. As the computational demands of real-time embedded systems continue to grow, effective yet transparent energy-management approaches will become increasingly important to minimize energy consumption, extend battery life, and reduce thermal effects. We believe that energy-efficiency and scheduling of real-time systems are closely related problems, which should be tackled together for best results. By exploiting the characteristic parameters of real-time application tasks, the energy-consciousness of scheduling algorithms and the quality of service of real-time applications can be significantly improved. In the following, we provide our thesis statement.

Thesis Statement. *The goal of this dissertation is to ameliorate, through scheduling, the energy-efficiency of real-time systems that can be proven predictable and temporally correct over multiprocessor platforms. The proposed solution(s) should be flexible to varying system requirements, less complex, and effective. Achievement of this goal implies that battery-operated real-time systems can still meet timing constraints while minimizing energy consumption, extending battery life, and reducing thermal effects.*

To support our thesis, this dissertation proposes energy-aware scheduling solutions of complex real-time applications that are scheduled over multiprocessor architectures. In section 1.2, we provide an overview of each technical contribution presented in this dissertation. A detailed background on real-time and energy-aware systems and real-time scheduling is provided in chapter 2. Note that we review state-of-the-art related to our specific contributions in each chapter. However, related research work is also referred throughout the document where pertinent.

1.2 Contributions

Energy-efficiency in real-time systems is a multi-faceted optimization problem. For instance, energy optimization can be achieved at both hardware- and software-levels while designing the system and at scheduling-level while executing application tasks. Both the hardware and software are concerned and can play an important role in the resulting energy consumption of overall system. In this dissertation, we focus

on the software-based aspects, particularly scheduling-based energy-consciousness in real-time systems. We develop novel power and energy management techniques while taking into account the features offered by existing and futuristic platform architectures. In the following, we discuss specific contributions presented in each chapter of this dissertation.

Chapter 3. In this chapter, we present our first contribution which is a multiprocessor scheduling algorithm, called *Two-Level Hierarchical Scheduling Algorithm (2L-HiSA)*. This algorithm falls in the category of *restricted-migration* scheduling. The EDF scheduling algorithm has the least runtime complexity among job-level fixed-priority algorithms for scheduling tasks on multiprocessor architecture. However, EDF suffers from sub-optimality in multiprocessor systems. 2L-HiSA addresses the sub-optimality of EDF as global scheduling algorithm and divides the problem into a two-level hierarchy of schedulers. We have ensured that basic intrinsic properties of optimal single-processor EDF scheduling algorithm appear in two-level hierarchy of schedulers both at top-level scheduler as well as at local-level scheduler. 2L-HiSA partitions tasks statically onto processors by following the bin-packing approach, as long as schedulability of tasks partitioned on a particular processor is not violated. Tasks that can not be partitioned on any processor in the platform qualify as migrating or global tasks. Furthermore, it makes clusters of identical processors such that, per cluster, the unused fragmented computation power equivalent to at most one processor is available. We show that 2L-HiSA improves on the schedulability bound of EDF for multiprocessor systems and it is optimal for hard real-time tasks if a subset of tasks can be partitioned such that the under-utilization per cluster of processors remain less than or equal to the equivalent of one processor. Partitioning tasks on processors reduces scheduling related overheads such as context switch, preemptions, and migrations, which eventually help reducing overall energy consumption. The NP-hardness of partitioning problem [60], however, can often be a limiting factor. By using clusters of processors instead of considering individual processors, 2L-HiSA alleviates bin-packing limitations by effectively increasing *bin* sizes in comparison to *item* sizes. With a cluster of processors, it is much easier to obtain the unused processing power per cluster less than or equal to one processor. We provide simulation results to support our proposition.

Chapter 4. Our second contribution, presented in this chapter, is a dynamic power management technique for multiprocessor real-time systems, called *Assertive Dynamic Power Management (AsDPM)* technique. This technique works in conjunction with global EDF scheduling algorithm. It is an admission control technique for real-time tasks which decides when exactly a ready task shall execute. Without this admission control, all ready tasks are executed as soon as there are enough computing resources (processors) available in the system, leading to poor possibilities of putting some processors in power-efficient states. AsDPM technique differs from the existing DPM techniques in the way it exploits the idle time intervals. Conventional

DPM techniques can exploit idle intervals *only* once they occur on a processor –i.e., once an idle time interval is detected. Upon detecting idle time intervals, these techniques decide whether to transition target processor(s) to power-efficient state. AsDPM technique, on the other hand, aggressively extracts most of the idle time intervals from some processors and clusters them on some other processors of the platform to elongate the duration of idle time. Transitioning processors to suitable power-efficient state then becomes a matter of comparing idle time interval's length against the break-even time of target processor. Although, AsDPM is an online dynamic power management technique, its working principle can be used to determine static optimal architecture configurations (i.e., number of processors and their corresponding voltage-frequency level, which is required to meet real-time constraints in worst-case with minimum energy consumption) for target application through simulations. We demonstrate the use of AsDPM technique for both static and dynamic energy optimization in this chapter.

Chapter 5. This chapter presents our third contribution, which is an *inter-task* dynamic voltage and frequency scaling technique for real-time multiprocessor systems, called *Deterministic Stretch-to-Fit (DSF)* technique. The DSF technique is mainly intended for multiprocessor systems. Though, applying it on single-processor systems is also possible and in fact, rather trivial due to absence of migrating tasks. DSF comprises three algorithms, namely, *Dynamic Slack Reclamation (DSR)* algorithm, *Online Speculative speed adjustment Mechanism (OSM)*, and *m-Tasks Extension (m-TE)* algorithm. The DSR algorithm is the principle slack reclamation algorithm of DSF that assigns dynamic slack, produced by a precedent task, to the appropriate priority next ready task that would execute on the same processor. While using DSR, dynamic slack is not shared with other processors in the system. Rather, slack is fully consumed on the same processor by the task, to which it is once attributed. Such greedy allocation of slack allows the DSR algorithm to have large slowdown factor for scaling voltage and frequency for a single task, which eventually results in improved energy savings. The OSM and the m-TE algorithms are extensions of the DSR algorithm. The OSM algorithm is an online, adaptive, and speculative speed adjustment mechanism, which anticipates early completion of tasks and performs *aggressive* slowdown on processor speed. Apart from saving more energy as compared to the stand-alone DSR algorithm, OSM also helps to avoid radical changes in operating frequency and supply voltage, which results in reduced peak power consumption, which leads to an increase in battery life for portable embedded systems. The m-TE algorithm extends an already existing *One-Task Extension (OTE)* technique for single-processor systems onto multiprocessor systems. The DSF technique is generic in the sense that if a *feasible* schedule for a real-time target application exists under worst-case workload using (optimal or non-optimal) global scheduling algorithms, then the same schedule can be reproduced (using actual workload) with less power and energy consumption. Thus, DSF can work in conjunction with various scheduling algorithms. DSF is based on the

principle of following the *canonical* execution of tasks at runtime –i.e., an offline or static optimal schedule in which all jobs of tasks exhibit their worst-case execution time. A track of the execution of all tasks in static optimal schedule needs to be kept in order to follow it at runtime [10]. However, producing and keeping an entire canonical schedule offline is impractical in multiprocessor systems due to a priori unknown assignment of preemptive and migrating tasks to processors. Therefore, we propose a scheme to produce an *online canonical schedule* ahead of practical schedule, which *mimics* the canonical execution of tasks only for future m-tasks. This reduces scheduler's overhead at runtime as well as makes DSF an adaptive technique.

Chapter 6. While new energy management techniques are still developed to deal with specific set of operating conditions, recent research reports that both DPM and DVFS techniques often outperform each other when their operating conditions change [37, 20]. Thus, no single policy fits perfectly in all or most operating conditions. Our fourth and final contribution in this dissertation addresses this issue. We propose, in this chapter, a generic power and energy management scheme for multiprocessor real-time systems, called *Hybrid Power Management (HyPowMan)* scheme. This scheme serves as a top-level entity that, instead of designing new power/energy management policies (whether DPM or DVFS) for specific operating conditions, takes a set of well-known existing policies. Each policy in the selected policy set, when functions as a stand-alone policy, ensures deadline guarantees and performs well for a given set of operating conditions. At runtime, the best-performing policy for given workload is adapted by HyPowMan scheme through a machine-learning algorithm. This scheme can enhance the ability of portable embedded systems to adapt with changing workload (and platform configuration) by working with a larger set of operating conditions and gives overall performance and energy savings that are better than any single policy can offer.

Chapter 7. In this chapter, we provide general conclusions and remarks on our contributions and results. Moreover, we discuss some future research perspectives of this dissertation.

Appendixes. We provide two appendixes in this dissertation. Appendix A provides functional details on the simulation tool *STORM (Simulation TOol for Real-time Multiprocessor scheduling)* [108] that we use in our simulations throughout this dissertation. Appendix B provides some additional simulation results related to chapter 6.

1.3 Summary

As a result of contemporaneous evolution in the complexity and sophistication of real-time applications and multiprocessor platforms, the research on real-time sys-

tems has confronted with many emerging challenges. One such challenge that real-time research community is facing is to reduce power and energy consumption of these systems, while maintaining assurance that timing constraints will be met. As the computational demands of real-time systems continue to grow, effective yet transparent energy-management approaches are becoming increasingly important to minimize energy consumption, extend battery life, and reduce thermal effects. Power- and energy-efficiency and scheduling of real-time systems are closely related problems, which should be tackled together for best results. Our dissertation motivates this thesis and attempts to address together the problem of overall energy-awareness and scheduling of multiprocessor real-time systems. This dissertation proposes novel approaches for energy-management within the paradigm of energy-aware scheduling for soft and hard real-time applications, which are scheduled over identical multiprocessor platforms of type symmetric shared-memory multiprocessor (SMP). We believe that by exploiting the characteristic parameters of real-time application tasks, the energy-consciousness of scheduling algorithms and the quality of service of real-time applications can be significantly improved. Rest of this document provides our contributions in detail.

Chapter 2

Background on Real-time and Energy-efficient Systems

Contents

2.1 Real-time Systems

Real-time systems can be classified, based on the strictness of timing constraints, into three broad categories: hard real-time, soft real-time and firm real-time systems [42, 47, 77, 80, 105].

Hard real-time systems. In hard real-time systems, the completion of a correct operation after its deadline is considered as useless. Ultimately, this operation may cause a critical failure of the system or expose end-users to hazardous situations. In other words, the penalty for even a single temporal constraint violation is unacceptable in hard real-time systems. Aerospace, nuclear, power plant, and automobile applications would use such systems.

Soft real-time systems. Soft real-time systems lower their strictness of timing constraints as compared to hard real-time systems. In such systems, although it is still preferred to have operations completed before their deadlines, violation of timing constraints does not make produced outputs entirely useless or hazardous. Even if deadlines of most operations are missed, the system can continue to operate. Such systems are nonetheless referred to as *real-time* since they use real-time

mechanisms (such as real-time operating systems for instance) in order to meet as many deadlines as possible. Cellular phone and multimedia applications can use such systems as the consequences of missing deadlines could be smaller than the cost of meeting them in all possible circumstances.

Firm real-time systems. Firm real-time systems provide an intermediate paradigm between hard and soft real-time systems. In contrast to hard real-time systems, firm real-time systems tolerate some latency in operations –i.e., a deadline miss results only in a decreased quality of service. Basically, the notion of firm real-time is less strict than that of hard real-time since it allows deadlines to be missed, but it is more strict than soft real-time in the sense that only a predefined ratio of deadline miss is allowed. Systems such as flight ticketing data servers that require concurrency, but can afford a delay in seconds, may use firm real-time systems.

As mentioned in section 1.1, a real-time system must ensure that, by construction, it is provably temporally correct and predictable. A real-time system can be proven predictable and temporally correct by specifying the following three aspects. Firstly, the real-time workload –i.e., the computation produced by a real-time application that must complete prior to its deadline, is specified in the form of tasks. These tasks are often recurring in their nature in real-time systems. Secondly, the processing platform or hardware resources upon which the application tasks are executed. Thirdly, a scheduling algorithm that determines, at any time, which set of tasks execute on the processing platform. In the following, we discuss all three aspects in more detail.

2.1.1 Real-time Workload

Real-time applications have become more sophisticated and complex in their behavior and interaction over the time [3]. As mentioned in earlier section, an application is said to be *real-time* when it is subject to timing constraints for its individual jobs/events as well as for its overall system response. These timing constraints are usually applied by the system designer, however, they typically reflect a need for safety or sustainability of the system performance. Definition of these timing constraints categorize an application into hard real-time, soft real-time, or firm real-time applications. For instance, the ABS breaking system in cars and video streaming applications are good examples of hard and soft real-time systems, respectively. Typically, hard real-time applications work in closed and highly predictable environments. On the contrary, soft real-time applications execute in open and less predictable environments.

In real-time systems, a common assumption is that it is possible to decompose a real-time application into a finite set of discrete tasks. Each task represents certain functionality of application. These tasks possess certain characteristic parameters such as release instant, periodicity, deadline, and execution requirement. Based on these parameters, tasks may be specified according to different *task models*. A task

model is the format and rules for specifying a task system. Before elaborating different task models, we present these characteristic parameters, which are associated with real-time tasks. From now on, we say that a task T_i releases a job $T_{i,j}$ (where j is the index of the job) at time instant t to express the fact that T_i is *instantiated* exactly at instant t so that its treatment can be carried out. A job can therefore be seen as an *instance* of a task T_i. In the following, we recall classical definitions for certain parameters that characterize tasks of real-time applications.

Deadline (d_i) of a real-time task is one of the key parameters which reflects the timing constraint on its execution. This quantity can be expressed as a number of CPU clock cycles but other reference units can be used, such as CPU *ticks* for instance. Hereafter, we use the term *time unit* to refer to the used reference unit. In this dissertation, deadline will denote the *relative* deadline of T_i –i.e., relative to its last job release, with the interpretation that once the task releases a job, that job must be *completely* executed by d_i time units.

Period (P_i) of a real-time task in another key parameter which reflects the delay between two consecutive job releases of task T_i. This parameter can be interpreted in three distinct ways, each of which leads to a well-defined type of task. According to the interpretation given to the period, tasks can be classified into three categories of task models: periodic task model, sporadic task model, and aperiodic task model. We elaborate further these task models in section 2.1.1.1.

Note that, very often, the theoretical results proposed in the literature apply only to tasks that provide a particular relation between their period and deadline. Therefore, it is worth mentioning the specific vocabulary that characterizes such relations. T_i is said to be *constrained-deadline* task if $d_i \leq P_i$ or *implicit-deadline* task in the particular case where $d_i = P_i$. When the proposed result holds whatever the relation between period and deadline, T_i is said to be *arbitrary-deadline* task. Note that the following inclusion holds: an implicit-deadline task is a constrained-deadline task which is in turn an arbitrary-deadline task. Thanks to this inclusive relation between the task models, any property that holds for an arbitrary-deadline task also holds for a constrained- and implicit-deadline task.

Offset (O_i) refers to the time delay before the release time of the first job of a periodic real-time task. In other words, the offset corresponds to the release time of the first job $T_{i,1}$ of task T_i. When the whole application is modeled by a single set of tasks with identical offsets, the application is said to be *synchronous*; without loss of generality, the offset of every task can be considered as 0 and can be ignored. Otherwise, if the offsets for different tasks are not equal, the application is said to be *asynchronous*. Notice that the offset of a task is defined only if the task is periodic. This is because the release times of the jobs (including the first one) for sporadic/aperiodic tasks are not known beforehand.

Worst-case execution time (C_i) refers to the largest execution time needed to complete a distinct job of a task T_i, assuming that its execution is not interrupted. Since real-time systems are designed to achieve only a few specific functions on a specific processing platform, mostly it is assumed that the Worst-Case Execution Time (WCET) of every task is known beforehand. WCET of a task is usually expressed in the same units as deadline and period. Note that the value of C_i depends not only on the functional code of T_i, but varies on different platforms. Authors in [123] suggest that the estimated WCET of tasks must offer both *tightness* and *safety* properties. Tightness means that they must be as close as possible to the actual WCET of a task, not to overestimate the resources required by the system. Safety is the guarantee that the computed WCET is greater than or equal to any possible execution time. The process of determining C_i must account for issues like worst-case cache behavior, pipeline stalls, memory contention, memory access time, program structure, and worst-case execution paths within the code.

There are some other factors as well, which are not directly concerned with the estimation of WCET, but contribute to the response time of tasks such as job preemptions, context switching, state saving, and scheduling-decision processing time by operating system. If a job is allowed to migrate between processors during its scheduling window, there may be an added penalty of refreshing the cache of the processor, to which the job is migrating. The preemption and migration costs are typically dependent on the processor architecture and the scheduling algorithm.

Within the scope of this dissertation, we consider that the WCET of all tasks is known beforehand. Interested reader may refer to some recent research work and surveys presented in [123, 69, 70] for further investigations in this research field.

Laxity (L_i) is a runtime parameter of a task's job that is a measure of its urgency to execute relative to its deadline. For instance, in a feasible task set, a job with zero laxity is the most urgent job to execute in order to avoid deadline miss. The absolute laxity (L_i) of a task at its release time instant t is given by equation 2.1.

$$L_i = d_i - (t + C_i) \tag{2.1}$$

Figure 2.1 illustrates a sample schedule of tasks and graphical representation of various characteristic parameters. This figure illustrates the schedule of a single periodic task T_i having implicit deadline –i.e., d_i=P_i, on a single processor. The parameters of T_i are: O_i=2, C_i=3, and d_i=P_i=4. Each green box represents a job of task T_i and its length corresponds to its worst-case execution time C_i. The release and deadline time instants are represented by up and down arrows, respectively. According to the definitions above, T_i releases a job noted $T_{i,j}$ ($\forall j$, j=$1, 2, ..., \infty$) at each instant r_i. Each such job has a WCET of C_i and it must complete by its relative deadline noted d_i. At release instant, a task has an absolute laxity of L_i.

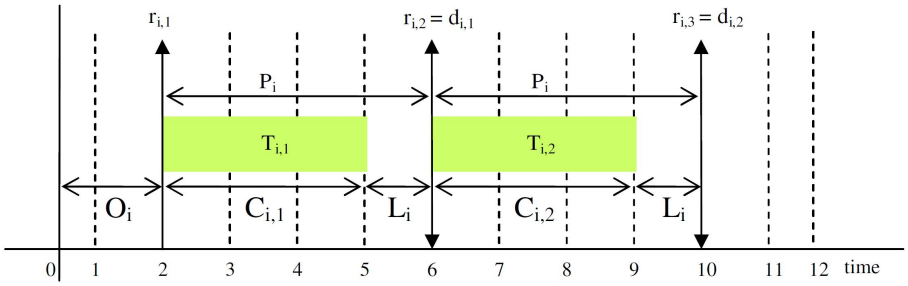


Figure 2.1: Illustration of various characteristic parameters of real-time tasks. Periodic task T_i has an implicit deadline (d_i=P_i) with the following values of other parameters. O_i=2, C_i=3, d_i=P_i=4, and L_i=1.

2.1.1.1 Task models

Based on the knowledge of characteristic parameters presented in section 2.1.1, we briefly discuss various classical task models in the following.

Periodic Task Model: This task model, presented by Liu and Layland in [71], allows the specification of homogeneous sets of jobs that recur at strict periodic interval. A periodic task T_i is specified by its offset, WCET, and period. Note that for such task models, every task has an *exact inter-arrival* time between successive jobs. Along with this interpretation, it is often assumed that the release time of the very first job of the tasks is also known beforehand, thus implying that the exact release time instants of every job can be computed at the system design-time.

Sporadic Task Model: The sporadic task model with implicit deadlines, presented by Liu and Layland in [71], removes the restrictive assumption of generating jobs at strict periodic intervals of time. In addition, an offset parameter is not specified for sporadic tasks. The behavior of a sporadic task T_i can be characterized by only the WCET and its period. The parameter P_i indicates the *minimum inter-arrival* time between successive jobs of T_i (note that P_i denoted the exact inter-arrival time for periodic tasks). That is, the exact release time of every job is not known before they are actually released at runtime.

Aperiodic Task Model: In this task model, the tasks do not have a *period* parameter. That is, system designers have no prior information about the time-instants at which jobs are released.

2.1.1.2 Description of workload model in this dissertation

Throughout this dissertation, we characterize a periodic and independent task set τ as a finite collection of tasks such that $\tau = \{T_1, T_2, T_i, ..., T_{n-1}, T_n\}$, and

a real-time task T_i composed of a finite or infinite collection of jobs such that $J = \{T_{i,1}, T_{i,2}, ...\}$. The letter n will denote the number of tasks in a task set. Every *job* $T_{i,j}$ of a real-time task $T_i, (\forall i, 1 \leq i \leq n)$ will be characterized by the quadruplet (r_i, C_i, d_i, P_i): an arrival or release time r_i, a worst-case execution requirement C_i, a relative deadline d_i, and a period P_i. The interpretation of these parameters is that the job $T_{i,j}$ of a task T_i arrives after r_i time units after the system start-time (the offset will be assumed zero in our general system model) and must execute for C_i time units over the time interval $[r_i, r_i + d_i)$. Release instant r_i is assumed to be a non-negative real number while both C_i and d_i are positive real numbers. The interval $[r_i, r_i + d_i)$ is referred to as $T_{i,j}$'s scheduling window. A job $T_{i,j}$ is said to be active at time instant t if $t \in [r_i, r_i + d_i)$ and $T_{i,j}$ has unfinished execution. In general task model, we consider a completely specified system –i.e., the system designer has complete knowledge of each job $T_{i,j}$ and infinitely-repeating jobs are generated by independent periodic tasks. We consider an implicit deadline task model. Furthermore, we consider that preemption of tasks –i.e., a job suspends while a different job executes and resumes execution at later time, is allowed. In all figures that illustrate scheduling of tasks throughout this dissertation, an upward arrow indicates a job's release and a downward arrow indicates its deadline and period. A rectangular box on the time line indicates that a task is executing during that interval as illustrated in figure 2.1.
When analyzing a system, we need to know the execution requirement of each task –i.e., the amortized amount of processing time the task will need. A task's *utilization* can be used to measure its processing requirement. The utilization of task T_i is the proportion of processing time the task will require if it is executed on a unit-speed (ν) processor: $u_i \stackrel{def}{=} C_i/P_i$. The aggregate utilization of a periodic task set, $U_{sum}(\tau) \stackrel{def}{=} \sum_{i=1}^{n} u_i$, measures the proportion of processor's time the entire task set will require.

In the rest of this dissertation, we consider that the worst-case execution time of tasks is known a priori, all jobs of tasks are preemptable, full migration of tasks is allowed (except in case of chapter 3), task-level parallelism is allowed, however, job-level parallelism is not permitted (i.e., a job may not execute concurrently with itself on multiple processors), and tasks are independent of each other –i.e., the execution of one task's job is not contingent upon the status of another task's job. Blocking of shared resources in not permitted as well.

2.1.2 Processing Platform

A complete real-time system is a real-time task model paired with a specific processing platform, which has a specific computing capacity. The platform may be composed of a *single* processor denoted by π or it may contain *multiple* processors denoted by Π such that $\Pi = \{\pi_1, \pi_2, ..., \pi_m\}$. Letter m refers to the number of processors in a multiprocessor platform. If the platform is a multiprocessor, the individual processors may all be the same (identical) or they may differ from one

another. As highlighted by authors in [3, 4], multiprocessor platforms are more energy efficient than equally powerful single-processor platforms, because raising the frequency of a single processor results in a *multiplicative* increase of the energy consumption while adding processors leads to an *additive* increase. More details are presented in section 2.2. In the following, we discuss certain categories of multiprocessor systems which differ from one another based on the speeds of the individual processors.

Unrelated multiprocessor platform. In these platforms, the processing speed depends not only on the processor, but also on the job being executed. In such platforms, a specific speed is associated to every processor-task couple with the interpretation that, in any time interval of length L, task T_i executes $\nu \times L$ execution units when executed on processor π_k. This model of platform was introduced in order to reflect the fact that two distinct tasks (i.e., with different code-instructions) executed on the same processor can require different execution times to complete even though the length of their code is identical. This is due to internal architecture of the processors and the type of the task instructions. Indeed, some processors are optimized for some types of instructions while they require more time to complete other types of instructions.

Uniform multiprocessor platform. In these platforms, the processing speed depends only on the processor. For instance, considering two different jobs, for all pairs of jobs $T_{i,j}$ and $T_{i,j+1}$ that execute on the same processor π_k, the processor speed remains the same.

Identical multiprocessor platform. In these platforms, all processors have the same speeds. Generally, in such systems, the speed is usually normalized to one unit of work per unit of time. The identical multiprocessor platform model considers that all the processors have the same characteristics, in term of power consumption, computational capabilities, architecture, cache size and speed, I/O and resource access, and access time to shared memory etc. In any interval of time, two identical processors execute the same amount of work and consume the same amount of energy.

In this dissertation, we consider an *identical multiprocessor* platforms for scheduling real-time tasks. Precisely, we consider *symmetric shared-memory multiprocessor (SMP)* layout of multiprocessor identical platform as illustrated in figure 2.2.

2.1.3 Real-time Scheduling

Real-time scheduling is one of the three aspects that should be taken into account to prove predictability and temporal correctness of real-time systems. The role of a real-time scheduling algorithm is to determine which active jobs of real-time

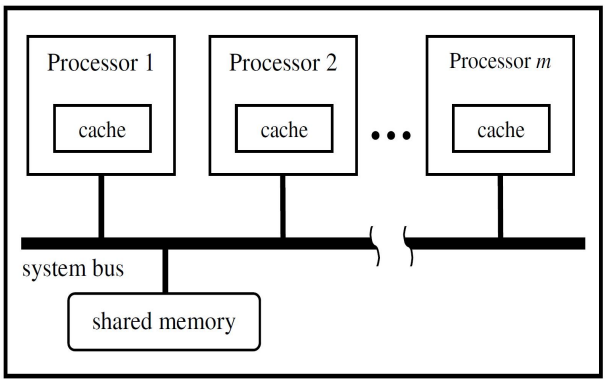


Figure 2.2: High-level illustration of symmetric share-memory multiprocessor (SMP) architecture layout of processing platform.

application tasks are executing on the processing platform at every time instant. From an abstract point of view, real-time scheduling algorithm determines the interleaving of execution for tasks' jobs on the target processing platform. This interleaving is called a *schedule*. The schedule must be produced to ensure that every job of task executes on processor(s) for its execution requirement (WCET) during its scheduling window. In a real-time schedule, generally, a task job can be in either *ready*, *running*, *blocked*, or *terminated* state. For instance, when a recurring job is released (for the first time as well as when it recurs), it becomes *ready* for execution. A ready job competes for its priority in the schedule with already present ready tasks. If a ready job has its priority high enough then it is allocated to a processor for execution and becomes *running*. A running job can be *blocked* due to the unavailability of a shared resource (other than the processor) held by another job[1]. Upon completion of its execution requirement (i.e., WCET), a job is said to be *terminated* until its next release. The instants at which a job of a task is released, preempted, terminated, or reached its deadline (for constrained-deadline task system) are broadly referred as *scheduling events*.

Scheduling algorithms can be broadly classified into *offline* and *online* algorithms [42, 47]. In offline scheduling algorithms, all scheduling decisions are made before the system begins executing. These scheduling algorithms select jobs to execute by referencing to a table describing the predetermined schedule. Usually, offline schedules are repeated after a specific time period. For instance, if the jobs being scheduled are generated by periodic tasks, an offline schedule may be generated for an interval of length equal to the least common multiple of the periods of the tasks (also referred as hyper-period) in the task set. After the hyper-period, the arrival pattern of the jobs will repeat. When the schedule reaches the end of

[1]Note that, in this dissertation, we do not consider inter-task dependency due to shared resources.

the predetermined table, it can simply return to the beginning of the table. In online scheduling algorithms, on the other hand, all scheduling decisions are made without specific knowledge of jobs that have not yet arrived. These scheduling algorithms select jobs to execute by examining properties of active jobs. Online algorithms can be more flexible than offline algorithms since they can schedule jobs whose behavior cannot be predicted ahead of time. Online scheduling algorithms can be divided into *fixed-priority* and *dynamic-priority* scheduling algorithms.

In fixed-priority scheduling algorithms, all jobs generated by the same task have the same priority. More formally, if job $T_{i,j}$ has higher priority than $T_{l,j}$ then $T_{i,j+1}$ has higher priority than $T_{l,j+1}$ for all values of j. Fixed-priority algorithms also referred as *Static-priority* algorithms. One very well-known fixed-priority scheduling algorithm is the *Rate Monotonic (RM)* algorithm proposed by [71]. In this algorithm, the task period is used to determine priority –i.e., tasks with shorter periods have higher priority. This algorithm is known to be optimal among *single-processor fixed-priority algorithms* –i.e., if it is possible for all jobs to meet their deadlines using a fixed priority algorithm, then they will meet their deadlines when scheduled using RM algorithm. In dynamic-priority scheduling algorithms, jobs generated by the same task may have different priorities. The *Earliest Deadline First (EDF)* algorithm [23, 71, 85] is a well-known dynamic-priority algorithm. EDF scheduling algorithm is optimal among *all single-processor scheduling algorithms* –i.e., if it is possible for all jobs to meet their deadlines, they will do so when scheduled using EDF. Dynamic-priority algorithms can be further divided into two categories –i.e., *job-level fixed-priority* and *job-level dynamic-priority* algorithms, depending on whether individual jobs can change priority while they are active. In job-level fixed-priority algorithms, jobs cannot change priorities. EDF is a job-level fixed-priority algorithm. On the other hand, in job-level dynamic-priority algorithms, jobs may change priority during execution. *Least Laxity First (LLF)* algorithm [33, 71] is a job-level dynamic-priority algorithm. LLF scheduling algorithm assigns a higher priority to a task with smaller laxity and it has been known as an optimal preemptive scheduling algorithm on a single processor platform.

Another important aspect of scheduling algorithms is their optimality. A scheduling algorithm is said to be *optimal* if it can successfully schedule any *feasible* task system. A task system is said to be feasible if it is guaranteed that a schedule exists that meets all deadlines of all jobs, for all sequences of jobs that can be generated by the task system. For instance, EDF is an optimal scheduling algorithm for single-processor systems [85] whereas, Rate monotonic (RM) is not an optimal algorithm for all single-processors. RM is optimal on single-processor systems only among fixed-priority algorithms –i.e., if it is possible for a task set to meet all deadlines using a fixed-priority algorithm then that task set is RM-schedulable [85]. Authors in [71] proved that for a set of n periodic tasks with unique periods, a feasible schedule that will always meet deadlines exists if the aggregate utilization of tasks is below a specific bound depending on the number of tasks –i.e., $\sum_{i=1}^{n} u_i \leq n\left(\sqrt[n]{2}-1\right)$.

When the number of tasks tends towards infinity, the schedulable utilization of RM algorithms tends towards a constant value –i.e., $n \to \infty; \sum_{i=1}^{n} u_i \approx 0.693$. Authors in [21] propose a hyperbolic bound relative to the Liu and Layland bound [71] and show that for n tending to infinity, the hyperbolic bound was found to be equal to $\sqrt{2}$. Single-processor systems that allow dynamic-priority scheduling will commonly use the EDF scheduling algorithm, while systems that can only use fixed-priority scheduling algorithms will use the RM scheduling algorithm. Since the focus of this dissertation is mainly on multiprocessor real-time systems, therefore, we discuss in the following how scheduling problem in multiprocessor systems is addressed.

2.1.4 Real-time Scheduling in Multiprocessor Systems

In multiprocessor systems, the problem of scheduling tasks is typically solved using different approaches based on how much *migration* the system allows at runtime. A task is said to be *migrating* if its successive jobs (or parts of the same job) are executed on different processors. Based on the amount of allowable migration, three types of migration strategies can be considered [8, 42, 47, 63].

No Migration. In this type of scheduling strategies, tasks can never migrate. Each task is statically assigned to a specific processor before execution and, at runtime, all job instances generated by a task execute on the processor to which, the task is assigned. Figure 2.3 illustrates a partitioned scheduler in which, every processor maintains a unique priority space associated only with the tasks being partitioned on it. No migration strategies are also referred as *partitioned scheduling* strategies. Partitioned scheduling approach has the virtue of permitting schedulability of task set to be verified using well-established single-processor schedulability analysis techniques.

Full Migration. In this type of scheduling strategies, jobs of a task can migrate at any point in time during their execution. All jobs are permitted to execute on any processor of the system. However, a job can only execute on at most one processor at a time –i.e., job parallelism is not permitted. Figure 2.4 illustrates a full migration scheduling in which, a single priority space is associated with all processors in the system. Full migration strategies are also referred as *global scheduling* strategies.

Restricted Migration. In this type of scheduling strategies, tasks can migrate only at job boundaries. Whenever a new job of a task is released, a top-level scheduler assigns this job to a particular processor. Once assigned, this job must complete its execution on the processor to which it is assigned –i.e., it can not migrate. However, the next job of the same task can execute on the same or different processor. Once assigned, the execution of job is the responsibility of the local scheduler on that processor. Figure 2.5 illustrates a restricted migration scheduler in which, there is a global priority queue and local priority queues for each processor.

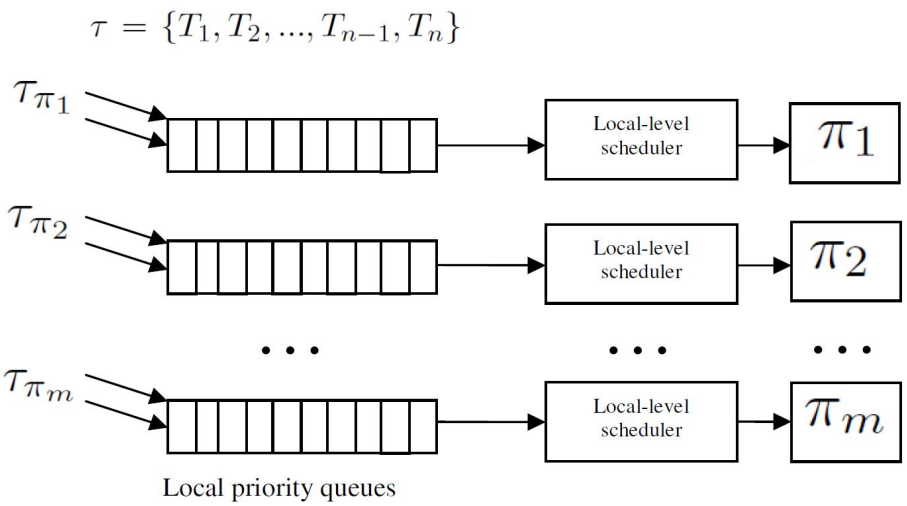


Figure 2.3: No migration scheduling.

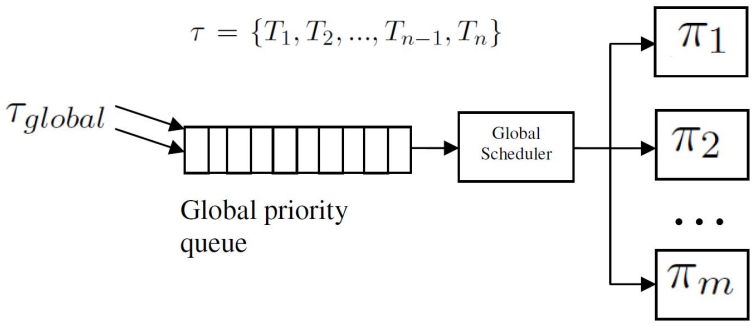


Figure 2.4: Full migration scheduling.

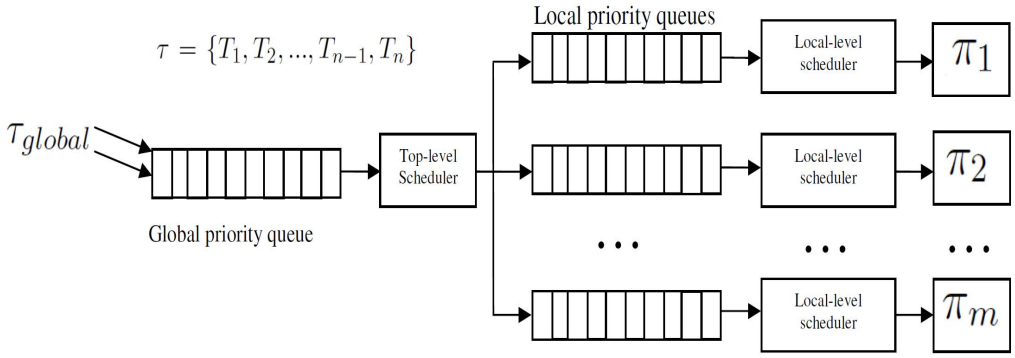


Figure 2.5: Restricted migration scheduling.

Prohibiting migration, as in case of partitioned scheduling, may cause a system to be under-utilized [8, 47] and for that reason, more than enough processing power will be available on some processor when a new job arrives. If migration is allowed, on the other hand, the job can execute for some time on one processor and then move to another processor, allowing the spare processing power to be distributed among all the processors. However, while full migration strategy is the most flexible, there are clearly overheads associated with allowing migration such as increased context switching, handling of shared resources, and cache-related overhead etc. Thus, there is a trade-off between scheduling loss due to migration and scheduling loss due to prohibiting migration.

2.1.4.1 Earliest Deadline First (EDF) as multiprocessor real-time scheduling algorithm

In this dissertation, EDF scheduling algorithm is often used to schedule real-time tasks on multiprocessor platform. In this section, we present EDF as a multiprocessor real-time scheduling algorithm. EDF is a job-level fixed-priority online scheduling algorithm which is optimal for single-processor systems [23, 71]. Authors in [34, 52] have shown that, for multiprocessor systems, there is no job-level fixed-priority optimal online scheduling algorithm. Since EDF is a job-level fixed-priority scheduling algorithm for multiprocessors, determining whether a given task set is feasible on a multiprocessor platform will not tell us whether that task set is *EDF-schedulable* on the same platform as well. Thus, EDF is not optimal for multiprocessor systems. Nonetheless, there are still many compelling reasons for using EDF for scheduling real-time applications on multiprocessor systems.

- Since EDF is an optimal single-processor scheduling algorithm, therefore, all *local* scheduling decisions are taken using an optimal algorithm when EDF is used in partitioning and restricted-migration based systems.
- EDF is considered as efficient from implementations point of view [74].
- The number of preemptions and migrations incurred by EDF can be bounded. Bounds depend on which migration strategy is being used. Since migration and preemption both incur overheads, it is important to be able to incorporate the overheads into any system analysis. This can only be done if the associated overheads can be bounded [52, 47].

On single-processor, EDF is well defined –i.e., for execution at every time instant, the job that has the smallest deadline is selected for execution on the sole processor. EDF is optimal scheduling algorithm for single-processo systems. When more processors are added to the system, however, EDF suffers from sub-optimality. The utilization bound for periodic tasks with implicit deadlines under EDF multiprocessor scheduling algorithm cannot be higher than $\frac{(m+1)}{2}$ for an m-processor platform [7]. This is a sufficient condition bound. One of the contributions of this

dissertation (in chapter 3) is to increase this schedulability bound of EDF algorithm using restricted-migration strategy.

2.2 Power- and Energy-efficiency in Real-time Systems

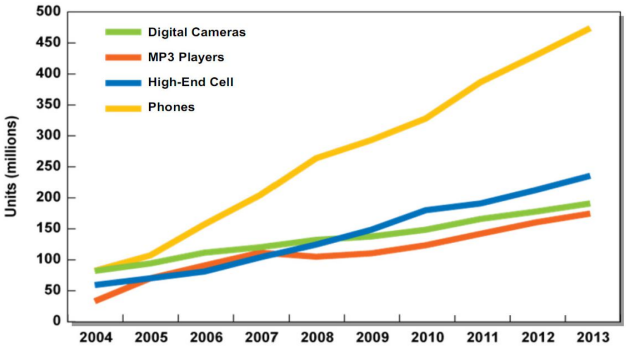
The demand for portable systems is ever-increasing with more complex functionality requirements as depicted in figure 2.6(a). The assessment of ITRS (International Technology Road map for Semiconductors) in 2008 fore casted further increase in power consumption in cell phones (see figure 2.6(b)). EPoSS (the European Technology Platform on Smart Systems Integration) suggested in 2009 that the energy density in batteries would increase beyond 400 Wh/kg by 2020 (see figure 2.6(c)). All these assessments suggest that complex real-time systems, which are composed of sophisticated real-time applications being scheduled over multiprocessor platforms, must be increasingly challenged to reduce energy consumption while maintaining assurance that timing constraints will be met. Power and energy in these complex systems is managed at both system design-time as well as runtime. In a post-design scenario, energy saving is achieved by static (offline) optimizations as well as by actively changing the power consumption profile of the system at runtime (online).

2.2.1 Power and Energy Model

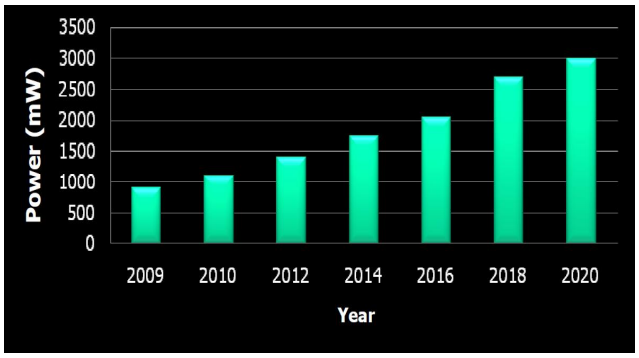
There are two principle sources of power dissipation in CMOS (Complementary Metal-Oxide Semiconductor) technology-based systems: *dynamic power* dissipation, which arises from the repeated capacitance charge and discharge on the output of the hundreds of millions of gates in modern chips, and *static power* dissipation which arises from the electric current that leaks through transistors even when they are turned off. Until very recently, only dynamic power dissipation has been a significant source of power consumption. However, shrinking processor technology below 100 nanometer has allowed and actually required reducing the supply voltage. Reduced feature-size favors dynamic power dissipation but unfortunately, smaller geometries exacerbate leakage, so static power begins to dominate the power consumption in deep sub-micron technology. Overall power consumption of CMOS technology-based processors, represented as a function of speed (ν) in variable speed settings, is composed of static and dynamic components which relate to supply voltage V_{op}, operating frequency F_{op}, and leakage current ($\mathrm{I}q$) through an approximate relation given by equation 2.2.

$$Pwr(\nu) = \gamma C_{eff} V_{op}^2 F_{op} + I_q V_{op} \tag{2.2}$$

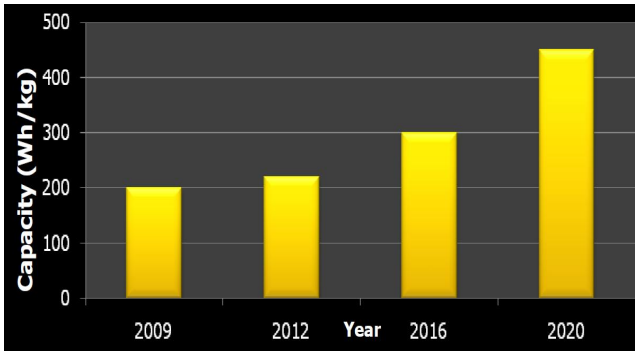
Here, γ is the fraction of gates actively switching and C_{eff} refers to the total load capacitance of all gates. The first addend in equation 2.2 corresponds to dynamically dissipated power and second addend models statically dissipated power. We have ignored power lost to the momentary short circuit that occurs at the output whenever the switching activity takes place. The loss is relatively small; it



(a) Evolution of the demand for portable equipment over the years (SEMICO Research Corp.)



(b) Power consumption in portable equipment over the years (ITRS 2008).



(c) Evolution of energy-density in batteries over the years (EPoSS 2009).

Figure 2.6: Current and future trends in the evolution of portable embedded system demand, their power consumption, and their energy-density in batteries. (a) Evolution of the demand for portable equipment over the years (SEMICO Research Corp.). (b) Power consumption in portable equipment over the years (ITRS 2008). (c) Evolution of energy-density in batteries over the years (EPoSS 2009).

contributes to dynamic power dissipation, and the first term in equation 2.2 can absorb it, if necessary.

When dynamic power is the dominant source of power consumption –as it has been historically in many less aggressive fabrication technologies– it is possible to approximate equation 2.2 with just the first term. In that case, the relation of V_{op}^2 suggests reducing supply voltage as the most effective way to decrease power consumption. For instance, halving the voltage will reduce the power consumption by a factor of four. Supply voltage is also related to the operating frequency of the processors by the relationship given in equation 2.3.

$$F_{op} \approx \frac{(V_{op} - V_{th})^{\eta}}{V_{op}} \tag{2.3}$$

Where V_{th} is the threshold or switching voltage and the exponent η is an experimentally derived constant that depends on the technology in use. During the past decades, the threshold voltage of the manufactured devices was too high to generate a significant leakage current when the state of the device is off, but still low enough compared to V_{op} to be ignored in the above expression. That is, the expression above could be rewritten as in equation 2.4.

$$F_{op} \approx (V_{op})^{\eta-1} \tag{2.4}$$

For instance, in technology based on classical MOSFETs (Metal-Oxide Semiconductor Field Effect Transistor), $\eta = 2$ [80], making the frequency a linear function of the supply voltage. However, the exact knowledge of η is not essential. The most important feature is the fact that the power function $Pwr(\nu)$ is a strictly increasing convex function of the frequency. Historically, CMOS technology has dissipated much less power. In fact, when not switching, CMOS transistors lost negligible (static) power. However, the power they consume has increased dramatically with increases in device speed and chip density. Continuously shrinking transistor size have forced a reduction of the threshold voltage as well. This miniaturization reduces the gap between the supply voltage and threshold voltage, resulting in a significant sub-threshold leakage current. Nowadays, these leakage currents are becoming a significant factor to portable devices because of their undesirable effect on battery life time. Hence, static power dissipation can no more be ignored in modern embedded systems. In this dissertation, we consider that static power is a significant contributing factor to overall power and energy dissipation and cannot be ignored any further.

Although power-efficiency and energy-efficiency are often perceived as overlapping goals, there are certain differences when designing systems for one or the other. Formally, the energy consumed by a system is the amount of power dissipated during a certain period of time. For instance, if a task occupies a processor during an execution interval of $[t_1, t_2]$ then the energy consumed by the processor during this time interval is given by equation 2.5.

$$E[t_1, t_2] = \int_{t_1}^{t_2} Pwr(\nu(t))dt \tag{2.5}$$

Equation 2.5 shows that there is an aspect of *time* involved in energy consumption of the system. Every computation operation requires a specific interval of time to be completed. The energy consumption decreases if the time required to perform such operation decreases and/or the power consumption decreases. For instance, power can be halved by simply halving the operating frequency, but at the same time, overall computation time would be doubled, which might be leading to no effect on overall energy consumption. Thus, a technique that would purely minimize power dissipation, but at the same moment increase the computational time, might lead to non change or even an increase in energy consumption.

Power- and energy-efficiency and scheduling of real-time systems are therefore closely related problems, which should be tackled together for best results. This dissertation is an attempt to address together the problem of overall energy-awareness and scheduling of multiprocessor real-time systems.

2.2.2 Energy-aware Real-time Scheduling

To address the issue of energy consumption, many scheduling-based software techniques have been proposed over the years, e.g., [16, 17, 57, 79, 83, 92, 124]. Energy-efficient scheduling techniques can be broadly classified into *online* and *offline* techniques.

In the category of online power and energy management techniques, *Dynamic Power Management (DPM)* technique is well studied and practiced in real-time systems. This technique selectively puts system components into power-efficient states whenever they are idle due to unavailability of workload. The fundamental theory for the applicability of DPM techniques is that systems (and their components) experience nonuniform workloads during operation time and that it is possible to predict, with a certain degree of confidence, the fluctuations of workload [102]. Hence, based on these predictions, DPM encompasses a set of techniques that achieve energy-efficient computation by selectively turning-off or reducing the performance of system components when they are idle or partially unexploited, hence conserving power. However, the inconvenience with DPM techniques is that once in a power-efficient state, bringing a component back to the active or running state requires additional energy and/or latency to serve an incoming task. Once applied, DPM policies eliminate both dynamic as well as static power dissipation. The input to the problem of managing energy consumption under DPM techniques is the length of an upcoming idle period, and the decision to be made is whether to transition system components to a power-efficient state while the system is idle. There are several issues in coming to this decision intelligently. For instance, immediate shutdown –shutdown as soon as an idle period is detected– may not save overall energy if the idle period is so short that the powering-up costs are greater than the energy saved in the sleep state. On the other hand, waiting too long to power-down may not achieve the best-possible

energy reductions either. Thus, there exists a need for effective and efficient decision procedures to manage power consumption.

Dynamic power management attempts to make such decisions (usually, under the control of scheduling algorithms) at runtime based on the dynamically changing system state, functionality, and timing requirements [29, 55, 92, 100]. Figure 2.7 illustrates a simple example of how a DPM technique takes energy management decision under the control of the scheduler. Upon the termination of a precedent task T_i, an idle time interval of length T_{idle} is detected on processor π. A DPM technique would compare the length of idle interval with the *break-event time (BET)* of π. For system components associated with non-zero transition costs, break-even time denotes the minimum length of idle interval which justifies (in terms of energy consumption) a device's transition from active state to power-efficient state [35]. A minimum value of BET is the one during which keeping a device in active state consumes exactly the same amount of energy as transitioning it from active to some other power-efficient state and bringing it back to active state. If $T_{idle} \geq BET$ then processor is transitioned to power-efficient state. Transition penalty in terms of time is highlighted as red boxes in figure 2.7. DPM techniques are further discussed in chapter 4.

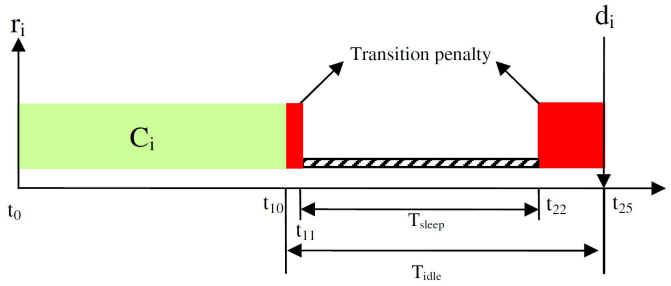


Figure 2.7: Example of energy management decision-making of DPM technique.

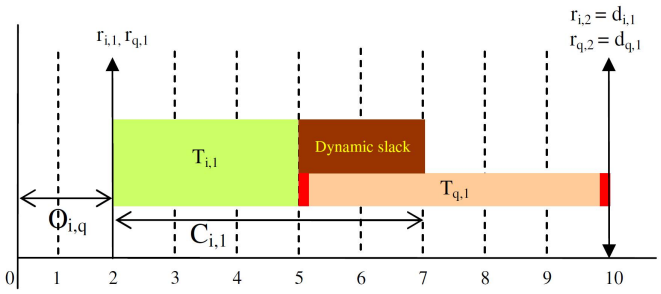


Figure 2.8: Example of energy management decision-making of DVFS technique.

Real-time *Dynamic Voltage and Frequency Scaling (DVFS)* technique is another

online technique which is aimed at changing the system's energy consumption profile. Real-time applications potentially exhibit variations in their actual execution time and therefore, often finish much earlier than their estimated worst-case execution time [10, 40]. DVFS technique exploits these variations in actual workload for dynamically adjusting the voltage and frequency of processors in order to reduce power and energy consumption. The challenge for these techniques, however, is to preserve the feasibility of schedule and provide deadline guarantees. These techniques are of particular effectiveness and interest because energy consumption of the processor is quadratically related to the supply voltage [87, 48]. Figure 2.8 illustrates a simple example of how a DVFS technique takes energy management decision under the control of scheduler. Figure 2.8 depicts two jobs –i.e., $T_{i,1}$ and $T_{q,1}$, of different tasks being released at the same time instant with exactly the same offset and equal deadlines. For equal deadlines, the scheduling algorithm can arbitrarily select any job in the absence of tie-breaking rules. Job $T_{i,1}$ executes first and finishes after executing only 3 time units which is earlier than its WCET –i.e., $C_{i,1}$=5 time units. Job $T_{i,1}$ generates 2 units of dynamic slack time (shown in red box between time instants $5-7$). Since $T_{q,1}$ is the only successor ready job left, therefore, it consumes this dynamic slack to slowdown its execution on target processor up to its deadline –i.e., time instant 10.

The key concern in a DVFS technique is to increase the utilization of slack time as much as possible and to make the resultant power and energy consumption as flat as possible by adjusting the operating frequency and supply voltage of a processor under real-time constraints. DVFS techniques can exploit not only the dynamic slack that is generated online by the workload variations, but also the worst-case (offline) slack time that occurs because of the under-utilization of processor, even if all tasks exhibit their worst-case execution times [68]. The worst-case slack time can be extracted from an application's schedule before task execution. DVFS techniques are further discussed in chapter 5.

2.3 Simulation Environment

For experimental results and evaluation of our proposed techniques in this dissertation, we rely mainly on the simulations using a multiprocessor simulation tool called *STORM*[2] *(Simulation TOol for Real-time Multiprocessor scheduling)* [108]. This tool has been initially designed and developed to satisfy the evaluation and validation needs of French national project PHERMA (Parallel Heterogeneous Energy efficient real-time Multiprocessor Architecture) [86]. STORM is intended to: i) use as input the specifications of the hardware and software architectures together with the scheduling policy; ii) simulate the system behavior using all the characteristics (task execution time, processor functioning conditions, etc.) in order to obtain the chronological track of all the scheduling events that occurred at run time, and iii) compute various real-time metrics in order to analyze the system behavior and

[2]STORM has been developed at IRCCyN laboratory of the University of Nantes, France [116].

performances from various point of views. Interested readers can see Appendix A for details on the functional aspects of STORM simulator.

We use H.264 video decoder application, which is a high compression rate multimedia application [88], and synthetic task sets as our target application model in this dissertation. While H.264 video decoder represents a computation extensive real-world multimedia application, the use of synthetic task sets allows us to vary task parameters as desired and observe the output behavior of our proposed techniques. These synthetic task sets are mostly auto-generated tasks. The criteria for generating synthetic tasks is presented in the experimental setup of chapters, where necessary. For processing platform, we use hardware parameters from Marvell's XScale® technology-based embedded processor PXA270 [72] to carry-out simulations. Although, PXA270 processor is not manufactured using most advanced technology[3], it is still a suitable choice. PXA270 supports six discrete voltage and frequency levels as shown in table 2.1, which allows static and dynamic voltage and frequency scaling. Moreover, it has five power-efficient states as shown in table 2.2, which allows dynamic power management. The power consumption parameters presented in table 2.1 and table 2.2 will be used in all our simulation results.

Table 2.1: Voltage-frequency levels of PXA270 processor

Parameter	Level1	Level2	Level3	Level4	Level5	Level6
Voltage	1.55	1.45	1.35	1.25	1.15	0.90
Frequency	624	520	416	312	208	104
Active Power	925	747	570	390	279	116
Idle Power	260	222	186	154	129	64

Table 2.2: Power-efficient states of PXA270 processor @ 624-MHz & 1.55-volts

States	Power(mWatts)	Recovery Time(ms)
Running	925	0
Idle	260	0.001
Standby	1.722	11.43
Sleep	0.163	136.65
Deep sleep	0.101	261.77

2.4 Summary

In this chapter, we provide the reader the background on real-time and energy-efficient systems. We discuss various models for real-time workload and characteristic parameters of real-time tasks, architecture of processing platforms, real-time single-processor and multiprocessor scheduling paradigms. Moreover, we discuss

[3]PXA270 is manufactured at 180nm technology. It supports ARMv5TE instruction set.

power- and energy-efficiency in real-time systems. We have provided power and energy models and simulation environment that we use throughout this dissertation. Moreover, we use periodic and independent task model of real-time applications that are scheduled upon identical multiprocessor platform of type SMP using mostly the full migration or global scheduling approach (except in chapter 3, where restricted-migration scheduling approach is used). We discuss in this chapter that energy-efficiency and scheduling of real-time systems are closely related problems, which should be tackled together for best results. To support this thesis, we discuss how techniques that would purely minimize power dissipation can increase the computational time and eventually lead to no change or even an increase in energy consumption. The inter-dependency of scheduling and energy-awareness of real-time systems serves as principle motivation for this dissertation.

Chapter 3

Two-level Hierarchical Scheduling Algorithm for Multiprocessor Systems

Contents

3.1 Introduction

In section 2.1.4, we have seen that the design space of preemptive real-time multiprocessor scheduling algorithms can be categorized into full-migration, restricted-migration, and partitioned scheduling strategies based on the allowable migration in the system. In this chapter, we focus mainly on restricted-migration scheduling strategies (recall: tasks are allowed to migrate at job-boundaries only) and we present a hierarchical scheduling algorithm for multiprocessor real-time systems.

Briefly looking at the full migration or global scheduling class of algorithms, they are attractive in the worst-case schedulability. Few multiprocessor global scheduling algorithms such as PFair [13], LLREF [28], and ASEDZL [77] are known to be

optimal. However, their scheduling overhead such as context switches and number of migrations and preemptions in often criticized to be too large. Systems that prohibit full migration, on the other hand, must use either partitioning or restricted-migration strategy. Between these two, the partitioning strategy is more commonly used in current systems, reason being that partitioning-based solutions can reduce the problem of multiprocessor scheduling into multiple single-processor scheduling problems. However, partitioning can only be used for fixed task sets. If tasks are allowed to dynamically join and leave the system, partitioning is not a viable strategy because a task joining the system may force the whole system to be repartitioned, thus forcing tasks to migrate. Determining a new partition is a *bin-packing* problem, which is strong NP-hard problem [60]. Thus, repartitioning dynamic task sets incurs too much overheads.

Restricted-migration scheduling strategies (also referred as *semi-partitioned* scheduling) provide a good compromise between the full migration and the partitioning strategies [24, 63, 62]. It is flexible enough to allow dynamic tasks to join the system at runtime, but it does not incur large migration overheads as compared to full-migration strategies. This strategy is particularly useful when consecutive jobs of a task do not share any data since all data passed to subsequent jobs would have to be migrated at job boundaries. Furthermore, the scheduler used as *top-level* scheduler (if a hierarchy of schedulers exist) in restricted-migration is much simpler than the full-migration global scheduler. The full migration global scheduler needs to maintain information about all active jobs in the system, whereas the top-level scheduler in restricted migration strategy makes a single decision about a job when it arrives and then passes the job to a local scheduler that maintains information about the job from that point forward. Restricted migration strategies offer relatively low scheduling overhead at runtime and they are potentially very interesting from the point of view of system performance and energy consumption. In this chapter, we propose a scheduling algorithm based on restricted migration strategy, called the *Two-level Hierarchical Scheduling Algorithm (2L-HiSA)*. Our proposed scheduling strategy uses *Earliest Deadline First (EDF)* scheduling algorithm in a hierarchical fashion at both top-level and local-level scheduler. Authors in [8] highlight that a significant disparity in schedulability exists between EDF-based scheduling algorithms and existing global optimal scheduling algorithms. This is unfortunate because EDF-based algorithms entail lower scheduling and task-migration overheads. In this work, we show that by using multiple instances of EDF scheduling algorithm at different levels of hierarchy, the wort-case schedulability bound of EDF can be improved.

3.2 Related Work

Some novel and promising techniques in the category of restricted-migration scheduling have been proposed very recently with the main objective of reducing the runtime overhead of scheduler and improving the schedulability and system utilization bound

for multiprocessor systems.

Kato et al. in [63] have presented a *semi-partitioned* scheduling algorithm for sporadic tasks with arbitrary deadlines on identical multiprocessor platforms. In this research work, authors propose to qualify a task as *migrating* task *only if* it is not possible to partition them on any processor of the platform. Thus, there are mostly partitioned tasks and few migrating tasks which are allowed to migrate from one processor to another *only once* per period. The main idea of this algorithm consists in using a *job-splitting* strategy for migrating tasks. In terms of utilization share, a migrating task is *split* into more than one processor. A task is split in such a way that a processor is filled to capacity by the portion of the task assigned to that processor. However, only the last processor to which the portion is assigned may not be filled to capacity. Figure 3.1 illustrates a migrating task is executed exclusively among processors by splitting the deadline of each migrating task into the same number of windows as the processors across which the task is qualified to migrate. In figure 3.1, a migrating task T_k is split across the three processors. Task T_k is presumed to be executed within these *fixed* windows with *pseudo-deadlines* which are smaller than the actual deadline of task. Fixing such pseudo-deadlines with limited allowable migration makes system much less flexible as the migrating tasks must execute within these fixed time slots. Systems with fixed time windows can not take full advantage of early completion of real-time tasks and consequently, cannot apply aggressive energy management techniques. Moreover, the job-splitting may still lead to prohibitive runtime overheads for the system.

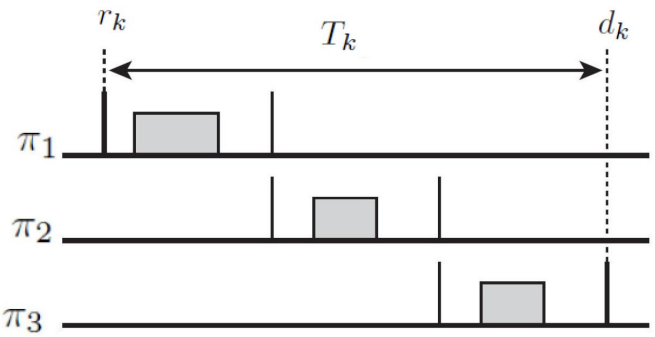


Figure 3.1: Job-splitting of a migrating task over three processors.

Authors in [62] have presented a *Earliest Deadline Deferrable Portion (EDDP)* algorithm, which is based on the portioned scheduling technique as well. Migrating tasks in this case are permitted to migrate between any *two* particular processors. In order to curb the cost of task migrations, EDDP makes at most $(m-1)$ migrating tasks on m-processors. Authors in this work claim that no tasks ever miss deadlines, if the system utilization does not exceed 65% using EDDP. The approach of limiting the migration of tasks to at most two processors is used earlier as well by authors in [8] who have proposed a scheduling algorithm which considers the

trade-off between system utilization and number of preemptions for recurring task systems. The migration overhead is relaxed in this approach compared to the other optimal multiprocessor algorithms by limiting the number of migrating tasks. The algorithm trades an achievable system utilization with the cost of preemptions by adjusting a parameter k, where $2 \leq k \leq m$. For $k < m$, the achievable utilization is claimed to be $k/(k+1)$. For $k = m$, on the other hand, it is 100%, thereby their proposed algorithm performs optimally. Based on the work of [8, 62, 63], authors in [38] have also propose a semi-partitioned hard real-time scheduling approach for sporadic deadline-constrained tasks upon identical multiprocessor platforms. In this work, migration of jobs is prohibited except that *two subsequent jobs* of a task can be assigned to different processors by applying a *periodic strategy*. This technique comprises two steps: an *assigning phase* and a *scheduling phase*. The assigning phase is somewhat similar to that of [63]. That is, if it is not possible to partition a task without violating schedulability guarantees then the concerned task is classified as *migrating* task. Authors propose to distribute jobs of migrating task among several processors using a *multi-frame tasking* approach with a predefined periodic sequence of the occurrence of jobs on various processors. This predefined sequence of jobs repeats itself cyclically at runtime upon the selected processors. The limitation of this approach is the assumption that the number of frames of each migrating task over multiple processors must be available *beforehand* to provide schedulability analysis. Moreover, in [38] and [62], the schedulability bound is 65% which is still not considerably large w.r.t. previously proposed partitioned scheduling algorithms offering 50% utilization bound in worst-case.

Calandrino et al. in [24] have proposed a *hybrid scheduling* approach for soft real-time tasks on large-scale multiprocessor platforms with hierarchical shared caches. In this approach, a multiprocessor platform is partitioned into *clusters*, tasks are statically assigned to these clusters (rather than individual processors), and scheduled within each cluster using the preemptive global EDF scheduling algorithm. All tasks are allowed to migrate *within* a cluster but not *across* clusters. Authors in this work demonstrate that, by partitioning the system into clusters instead of individual cores, bin-packing limitations can be alleviated by effectively increasing *bin-sizes* in comparison to *item-sizes*. However, this work still uses a common global scheduler at cluster-level which is equivalent to breaking a larger multiprocessor scheduling problem into *multiple smaller* multiprocessor scheduling problems. Moreover, the solution is limited to soft real-time applications. In contrast to [24], authors in [114] have proposed a *two-level* scheduling scheme, which uses the idea of *sporadic servers*. In this approach, first an application is partitioned into parallel tasks as much as possible. Then the parallel tasks are dispatched to different processors, so as to execute in parallel. On each processor, real-time tasks run concurrently with non real-time tasks. At the top level, a sporadic server is assigned to each scheduling policy while at the bottom level, a Rate-Monotonic (RM) scheduler is adopted to maintain and schedule the top-level sporadic servers. While this research work uses a two-level hierarchy of schedulers, only soft real-time applications are considered for scheduling.

Our proposed two-level hierarchical scheduling algorithm statically partitions as much tasks as possible to processors, which is somewhat similar to that of [62]. However, neither the number of *migrating* tasks nor the number of *migrations* per migrating task is limited in our approach, which is contrary to that of [24], [38], and [62]. Moreover, unlike in [62], 2L-HiSA does not *fix* time slots for migrating tasks. Rather it reserves a portion of processor time on each processor (in proportion to its under-utilization) for migrating tasks and this portion of time can be dynamically relocated by local-level scheduler within a specified period to allow the execution of statically partitioned tasks. This dynamic relocation of reserved time for migrating tasks improves system flexibility both at design-time and runtime. In section 3.3, we provide the 2L-HiSA scheduling algorithm in detail.

3.3 Two-level Hierarchical Scheduling Algorithm

The 2L-HiSA scheduling algorithm uses *multiple instances* of single-processor optimal EDF scheduling algorithm in a hierarchical fashion at two levels: an instance at top-level scheduler and an instance at local-level scheduler on every processor of the platform. Since EDF is an optimal single-processor scheduling algorithm, therefore, in order to determine whether the given task set is *EDF-schedulable*, it suffices to determine whether this task set is *feasible* on the single-processor systems. Unfortunately, it has been shown in [34, 52] that there are no optimal *job-level fixed-priority* scheduling algorithms for *multiprocessors*. Since EDF falls in this category, therefore, determining whether a given task set is feasible on a multiprocessor system will not tell us whether the same task set is EDF-schedulable on the same system as well. Baruah, et al. proved in [13] that there exists a *job-level dynamic-priority* scheduling algorithm, referred as PFair, which is optimal for periodic task sets on multiprocessors. Srinivasan and Anderson later showed in [106] that this algorithm can be modified to be optimal for sporadic task sets as well. However, these results do not apply on EDF because they use a job-level dynamic-priority algorithm. On the issue of determining EDF-schedulability, authors in [5] have provided schedulable utilization bounds for job-level fixed-priority scheduling algorithms for full-migration, restricted-migration, and partitioned scheduling strategies. EDF, being a job-level fixed-priority algorithm, has schedulable utilization bounds of $\frac{m^2}{2m-1} \leq U_{sum} \leq \frac{m+1}{2}$ for full-migration strategies, $U_{sum} = \frac{\beta m+1}{\beta+1}$ $(\beta = \lfloor \frac{1}{\alpha} \rfloor)$ for no-migration strategies, and $m - \alpha(m-1) \leq U_{sum} \leq \frac{m+1}{2}$ or otherwise for restricted-migration strategies, respectively. Here, the term α represents a cap on individual task utilizations. Note that, if such a cap is not exploited, then the upper bound on schedulable utilization is approximately $\frac{m}{2}$ or lower. Authors in [7] state that, for a periodic task set with implicit deadlines, the schedulable utilization under EDF or any other static-priority multiprocessor scheduling algorithm –partitioned or global– can not be higher than $\frac{m+1}{2}$ for m processors. Clearly, under this schedulability bound, a multiprocessor platform suffers heavily from under-utilization (i.e., by a factor of $\frac{m-1}{2}$). For instance, in a system composed of three processors ($m = 3$), platform re-

source equivalent to at least one processor $\left(\frac{m-1}{2} = 1\right)$ is wasted. 2L-HiSA, instead of using global EDF scheduling algorithm, proposes a hierarchical scheduling approach using multiple single-processor optimal EDF instances. Section 3.3.1 provides the basic concept of 2L-HiSA.

3.3.1 Basic Concept

The concept of two-level hierarchical scheduling algorithm slightly differs from the conventional restricted migration-based scheduling strategies. In restricted migration scheduling with hierarchical schedulers, all tasks can migrate at *job-boundaries* and they share a common top-level task queue as illustrated in figure 2.5 (chapter 2). That is, when a new job of a recurring task is released, the top-level scheduler assigns this job to any processor available in the platform. A released job, once assigned to a particular processor, can execute only on that processor under the control of local-level scheduler. Another job of the same task, however, can be assigned to a different processor. Thus, for every new job of a task, the top-level scheduler first decides its assignment to target processor in the platform and then the local scheduler executes that job according to its appropriate local priority level. In two-level hierarchical scheduling algorithm, however, local schedulers have certain number of partitioned tasks that do not migrate at all as in case of [62]. The 2L-HiSA algorithm is based on the concept of semi-partitioned scheduling, in which most tasks are statically assigned to specific processors, while a few tasks migrate across processors. Once partitioned, these tasks are entirely handled by local-level scheduler and always remain in unique priority space associated only to their respective processor as illustrated in figure 3.2 by τ_{π_1}, τ_{π_2}, and τ_{π_m}, respectively. A task is qualified to become *migrating* task only if it cannot be partitioned on any processor any more using simple bin-packing approach. Such tasks are *fully* migrating tasks, unlike the migrating tasks in case of [62] and [38], which limit the number of possible migrations per period or per processor. All migrating tasks are placed in a separate subset of tasks referred as τ_{global} as illustrated in figure 3.2. Only subset τ_{global} is handled by the top-level scheduler.

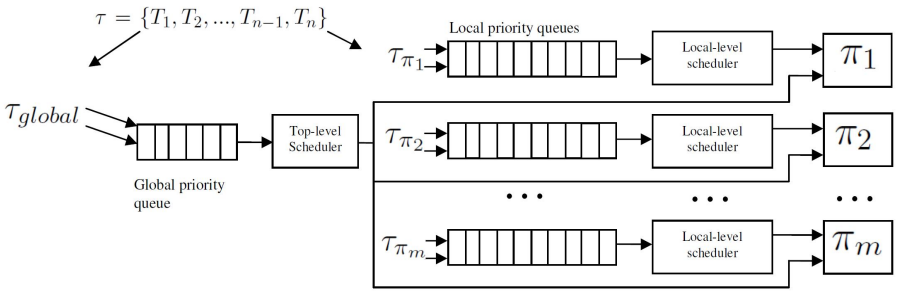


Figure 3.2: Two-level hierarchical scheduling approach based on restricted migration.

Top-level scheduler assigns tasks from τ_{global} to processors at runtime within *suitable* time slots. These suitable time slots are actually the portion of processor time that is determined on every processor based on its under-utilization (if any). Section 3.3.2 gives more details on the procedure for determining the size, periodicity, and priority of these time slots. However, it is worth mentioning here that these time slots occur *explicitly* on processors in an m-processor platform –i.e., they do not occur in parallel. Moreover, these time slots are not fixed (like in case of [62]) and occur dynamically *within* the specified period. Due to the NP-hardness of the partitioning problem, processors in a multiprocessor platform are often under-utilized with a significant margin in a post-partitioned scenario.

3.3.2 Working Principle

In this section, we provide the main steps of our proposed algorithm. 2L-HiSA, as mentioned earlier, is based on the concept of restricted migration scheduling and consists of two phases:

1. *The task-partitioning phase*: In this phase, each non-migrating task is (offline/statically) assigned to a specific processor by following the bin-packing approach.

2. *The processor-grouping phase*: This is a post-partitioning phase in which, processors are grouped together based on their workload characteristics.

3.3.2.1 The task-partitioning phase

Let us consider that a real-time task set τ containing at most n tasks such that $\tau = \{T_1, T_i, ...T_{n-1}, T_n\}$, has to be scheduled on an identical multiprocessor platform composed of m processors. The task set is considered feasible a priori -i.e., $U_{sum}(\tau) = \sum_{i=1}^{n} u_i \leq m$. In the first step of our algorithm, each task T_i is statically assigned to a particular processor π_k by following the bin-packing approach, as long as the task does not cause violation of schedulability of tasks being already partitioned upon processor π_k –i.e., $U_{sum}(\tau_{\pi_k}) \overset{def}{=}_{(L>0)} \frac{DBF(\tau_{\pi_k}, L)}{L} \leq 1$, where tasks being partitioned on a particular processor π_k are denoted by τ_{π_k}, L refers to interval length, and DBF refers to the classical Demand Bound Function [38, 77]. Note that the task-partitioning can be performed using any suitable partitioning strategy. Algorithm 1 illustrates the task partitioning phase. In the first step, before partitioning any task to processors, the utilization of each processor π_k is initialized to zero –i.e., $U_{\pi_k} = 0$ (lines $1-5$). In the second step, each task is tested for partitioning on m processors of the platform according to the condition mentioned earlier (lines $6-14$). If, for any task T_i, $U_{sum}(\tau_{\pi_k}) > 1$ –i.e., it can not be partitioned on π_k ($\forall k, 1 \leq k \leq m$), then this task is classified as *migrating* task and assigned to τ_{glob} subset of tasks (lines $1-18$).

For a feasible task set τ, often it is not possible to partition all tasks due to the NP-hardness of partitioning problem. Thus, in our algorithm, a given τ is divided

Algorithm 1 Offline task partitioning to processors

```
1: n ← number of tasks in τ
2: m ← number of processors in Π
3: for k = 1...m do
4:    U_{π_k} ← 0;
5: for i = 1...n do
6:    for k = 1...m do
7:       if U_{π_k} + u_i ≤ 1 then
8:          assign T_i to π_k;
9:          U_{π_k} = U_{π_k} + u_i;
10:         remove T_i from τ;
11:         break;
12: if size(τ) ≠ 0 then
13:    assign all remaining tasks to τ_glob;
```

into two subsets of tasks such that $U_{sum}(\tau_{part}) + U_{sum}(\tau_{glob}) = U_{sum}(\tau) \leq m$. In a post-partitioned scenario, we can calculate the aggregate utilization of tasks being statically partitioned on (or assigned to) every processor (π_k) individually using equation 3.1. Here, np refers to the total number of tasks being partitioned on a particular processor π_k.

$$U_{\pi_k}(\tau_{\pi_k}) = \sum_{i=1}^{np} \frac{C_i}{P_i} \quad (\forall i, 1 \leq i \leq np, \forall T_i \in \tau_{part}) \tag{3.1}$$

From equation 3.1, we can compute the under-utilization present on every processor π_k using equation 3.2. Let the under-utilization present on any processor π_k be referred as U'_{π_k}.

$$U'_{\pi_k}(\tau_{\pi_k}) = 1 - \sum_{i=1}^{np} \frac{C_i}{P_i} \quad (\forall i, 1 \leq i \leq np, \forall T_i \in \tau_{part}) \tag{3.2}$$

3.3.2.2 The processor-grouping phase

In the second step, we group processors of the platform Π in such a way that the cumulated under-utilization on all processors *within a group* is not greater than one –i.e., $\sum U'_{\pi_k} \leq 1$. In section 3.3.1, it is stated that a portion of processor time is reserved on every processor in proportion to its U'_{π_k} to which, the top-level scheduler could exploit for scheduling tasks from τ_{glob}. Moreover, these portions of processor time must appear explicitly. Now, if the cumulated under-utilization of processors will be more than one, then the computation power equivalent to more than one processor will be *free* within the system. This under-utilization will cause idle time intervals to appear in parallel on certain processors which is not desirable. Thus, grouping processors such that the sum of under-utilization on all processors *within a group* is not greater than one allows to have a cumulated (but still fragmented) computation power equivalent to at most one processor *free* within each group. This condition helps avoiding parallelism of the idle time intervals that would occur

due to under-utilization. Algorithm 2 illustrates processor-grouping phase. This algorithm outputs the number of possible processor-groups or clusters within the platform that respect above condition. However, limiting the amount of under-utilization per group is not the only condition to ensure explicit occurrence of idle time intervals. These idle intervals would still appear randomly within each group. An issue of concern here is, how to make the idle intervals *non-parallel* and *periodic* so that migrating tasks could consume them. In the following, we provide a simple illustrative example of how the idle time intervals would appear at runtime in an application's schedule under EDF algorithm and then we should answer the concern related to explicitness and periodicity of idle intervals required for 2L-HiSA.

Algorithm 2 Offline processor-grouping

1: $m \leftarrow$ number of processors in Π
2: $Y \leftarrow 0$; *//number of processor-groups*
3: $U^{'}_{sum} \leftarrow 0$;
4: **for** $k = 1...m$ **do**
5: $\quad U^{'}_{sum} \leftarrow U^{'}_{sum} + U^{'}_{\pi_k}$;
6: $\quad$ **if** $U^{'}_{sum} \geq 1$ **then**
7: $\quad\quad Y \leftarrow Y + 1$;
8: $\quad\quad U^{'}_{sum} \leftarrow 0$;
9: **output:** Y processor-groups are created;

Example 3.1: Let us consider a periodic task set τ composed of six tasks ($n = 6$) to be scheduled on a multiprocessor platform composed of four identical processors ($m = 4$). Task set τ is scheduled using EDF scheduling algorithm. The value of quadruplet of each task is selected such that $U_{sum}(\tau)$ respects sufficient condition bound provided by [7] –i.e., $U_{sum} \leq \frac{m+1}{2} \leq 2.5$. The values of quadruplet (r_i,C_i,d_i,P_i) are; τ=$\{T_1(0,3,7,7), T_2(0,7,14,14), T_3(0,5,11,11), T_4(0,4,13,13), T_5(0,4,8,8), T_6(0,3,9,9)\}$.

Let us partition these tasks on four processors[1] such that; τ_{π_1}=$\{T_4, T_6\}$, τ_{π_2}=$\{T_2\}$, τ_{π_3}=$\{T_5\}$, and τ_{π_k}=$\{T_1, T_3\}$. Figure 3.3 illustrates the EDF-schedule of τ on four processors. In this figure, it can be noticed that (more or less) every processor is under-utilized by a factor of $U^{'}_{\pi_k}(\tau_{\pi_k})$ as stated by equation 3.2. Moreover, due to this under-utilization, idle intervals appear on processors at random (based on the EDF scheduler's priority mechanism) and in a non-periodic fashion.

2L-HiSA aims at exploiting these random idle intervals to schedule tasks from τ_{glob} under the control of top-level scheduler. The problem, however, is that these idle intervals are not periodic in their occurrence and therefore, can not be used to schedule *periodic* tasks by the top-level scheduler. The intuitive idea behind the 2L-HiSA algorithm is to *force* these idle time slots to appear in a periodic fashion on all those processors which offer positive under-utilization such that the amount of *periodic* idle time should explicitly appear on each processor π_k in proportion to $U^{'}_{\pi_k}$ offered by that processor. Once idle time slots become periodic, tasks from τ_{glob} can then be placed in these time slots under the control of top-level scheduler. To achieve this objective, a *dummy task* is added on every

[1]The partitioning of tasks performed in this example may not be the optimal solution.

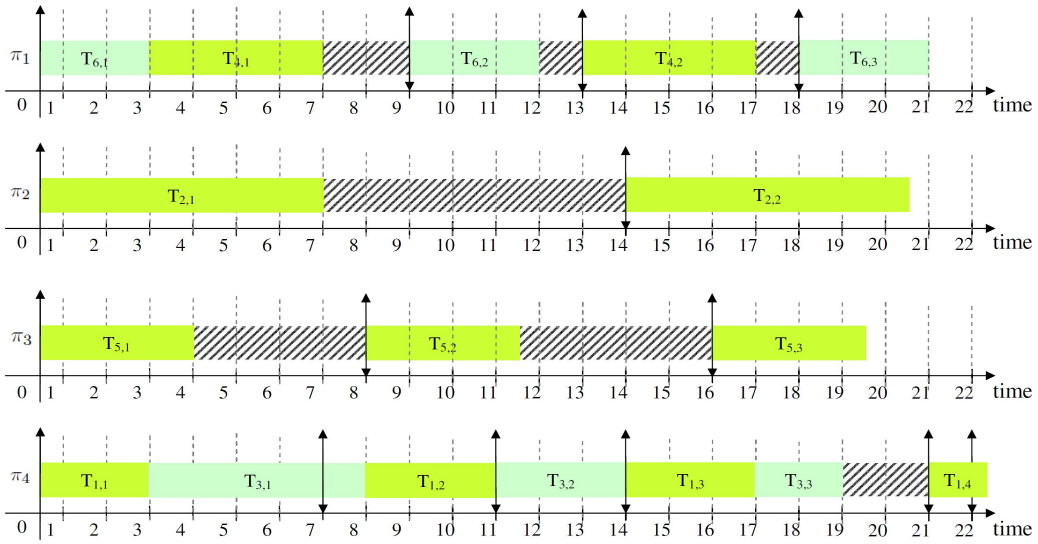


Figure 3.3: Example schedule of partitioned tasks under EDF scheduling algorithm on SMP architecture (n=6, m=4), illustrating the under-utilization of platform.

processor π_k. Let us call this dummy task as T_k^d on processor π_k. Task T_k^d is a periodic task that appears on all processors, which offer $U'_{\pi_k} > 0$. In the following, we calculate the parameters of T_k^d such as its period P_k^d and worst-case execution time C_k^d on every processor.

Period of T_k^d is selected as the *absolute minimum period* of all the tasks present in τ ($\tau_{part} \in \tau, \tau_{glob} \in \tau$) as shown in equation 3.3.

$$P_k^d = \min_{i=1}^{n} \{P_i\} \qquad (\forall k, 1 \leq k \leq m) \tag{3.3}$$

Note that, apart from being the smallest, the period of T_k^d is the *same* on all processors ($\forall k, 1 \leq k \leq m$). The advantage of having the smallest period for T_k^d on all processors is that the cumulated under-utilization $\sum U'_{\pi_k}$ present in a selected group of processors is proportionately available within the smallest period, hence, available for the most recurring migrating task. The advantage of having the same value for P_k^d on all processors is that it ensures the release of T_k^d at the same time on all processors, which is helpful in managing explicit execution of jobs of T_k^d on different processors. Once the period for T_k^d is determined, its worst-case execution time C_k^d can be calculated on every processor using equation 3.4, which is proportionate to U'_{π_k} available on each processor.

$$C_k^d = P_k^d \times U'_{\pi_k} \qquad (\forall k, 1 \leq k \leq m) \tag{3.4}$$

C_k^d refers to the *size* of idle time slots appearing on processor π_k at runtime over a period of P_k^d. Note that T_k^d is an *empty* task used only to reserve C_k^d time units of processor time over the smallest possible period P_k^d. At runtime, top-level scheduler *fills* these C_k^d time slots reserved by T_k^d with tasks from τ_{glob}. Since tasks in τ_{glob} are fully migrating tasks, therefore, they can use C_k^d time units on all processor if T_k^d does not appear in parallel.

Thus, one of the design consideration of 2L-HiSA is to make sure that T_k^d is non-parallel or explicit on processors within a group over the interval lengths of P_k^d.

Since T_k^d has the same period (P_k^d) on every processor, therefore, it releases at the same time on all processors. Moreover, making P_k^d being the smallest period within the task set τ also makes T_k^d the highest priority task on every processor under EDF scheduling algorithm. Thus, to ensure explicit execution of T_k^d, the 2L-HiSA algorithm performs a priority exchange of T_k^d with highest priority local task on all those processors within a group on which T_k^d is not selected for execution at time instant t. This priority exchange is non-blocking from the platform resources point of view –i.e., exchanging the priority of T_k^d with local/partitioned task on a processor π_k does not cause processor π_k to become idle or blocked as long as statically partitioned ready tasks exist. We illustrate this concept with an example in the following.

Let us consider a multiprocessor platform with three processors belonging to the same group. Each processor has a dummy task T_k^d assigned to it. At time instant $t = 0$, T_k^d is released on all three processors simultaneously. If T_1^d, which is the dummy task on processor π_1, is assigned on π_1 as illustrated in figure 3.4, then the local schedulers on π_2 and π_3 exchange the priority of T_2^d and T_3^d with local tasks, respectively, to give higher priority to statically partitioned ready task that is having the highest priority (if any). Upon termination of T_1^d on π_1 at time instant $t = 1$, remaining two processors π_2 and π_3 revert the priority of their respective dummy tasks T_2^d and T_3^d, respectively, to allow them to compete for priority at local scheduler's level. Since, at most one local scheduler can assign T_k^d on a processor at any time to ensure explicit execution of jobs of T_k^d, therefore, the other local schedulers exchange priority of their respective T_k^d again to allow partitioned tasks to execute. In this example, after T_1^d is terminated on π_1, local scheduler on π_2 assigns T_2^d and local scheduler on π_3 again exchanges the priority of T_3^d to let the partitioned tasks run. Finally, at time instant $t = 2$, local scheduler on π_3 assigns T_3^d for execution. At time $t = 3$, T_1^d, T_2^d, and T_3^d are released again and compete for assignment on their respective processors. Note that T_k^d has no specific order of occurrence –i.e., fixed time slot that can be defined a priori on different processors. A newly released job of T_k^d has to, first, compete for the priority among locally partitioned tasks and then compete for priority among T_k^d present on other processors within a group. Failure to obtain highest priority at any of the two levels cause a priority exchange for concerned task. This makes the portion of processors' time reserved for τ_{glob} to appear in a *sequential* fashion over P_k^d within a group of processors. The priority exchange for T_k^d on the same processor can be performed up to the time instant when laxity of T_k^d becomes zero.

3.3.3 Runtime View of Schedule from Different Levels of Hierarchy

In this section, we provide the reader the view-points of both top-level and local-level schedulers under the 2L-HiSA algorithm.

3.3.3.1 Local-level Scheduler

From the earlier discussion, we know that single-processor optimal EDF scheduling algorithm is used as local scheduler on every processor to schedule statically partitioned tasks. Along with tasks being partitioned on each processor π_k –i.e., τ_{π_k}, there is a dummy task T_k^d assigned on each processor that has an execution requirement of C_k^d, which is exactly equal to the amount of U'_{π_k} available on π_k –i.e., $U'_{\pi_k} = 1 - U_{\pi_k} = \frac{C_k^d}{P_k^d}$. Thus, the worst-case

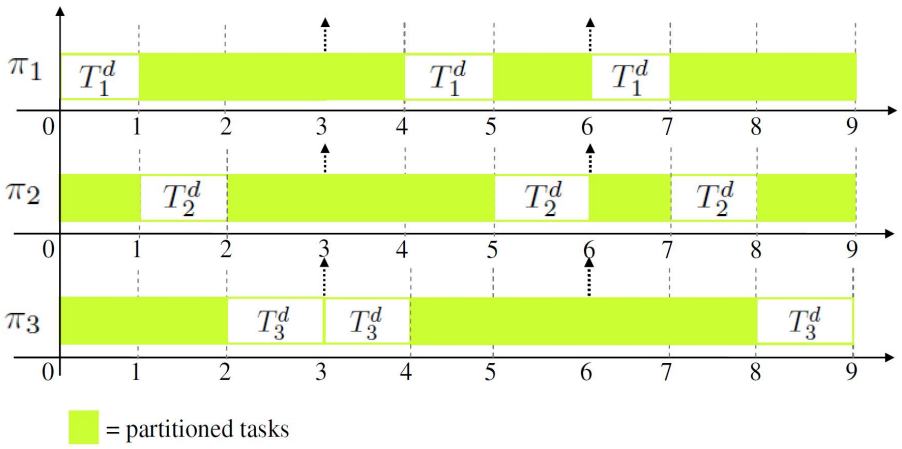


Figure 3.4: Illustration of T_k^d occurring on different processors with respect to the proportionate under-utilization available on each processor.

workload of each processor is maximum –i.e., $U_{\tau_{\pi_k}} + U'_{\pi_k} = 1$. Local EDF scheduler on each processor visualizes the problem of scheduling τ_{π_k} along with T_k^d reduced to single-processor scheduling problem for which, EDF is optimal. Algorithm 3 illustrates jobs assignment on processor by local-level scheduler. For each processor-group Y, the number of processors within each group are known a priori and dummy task T_k^d is added to the local priority queue (ReTQ) of each processor (lines $1-8$). Upon the arrival of a scheduling event, this ReTQ is sorted according to EDF priority and highest priority ready task is selected for execution (lines $9-11$). If the selected task is not T_k^d then it is directly assigned to processor π for execution (line 21). Otherwise, if selected task is T_k^d then local scheduler checks if T_k^d is already executing on any other processor in the system. If T_k^d is not assigned on any other processor then local scheduler schedules T_k^d on π. Otherwise, the priority of T_k^d is exchanged to allow subsequent higher-priority partitioned task from τ_{π_k} to execute on π.

Figure 3.4 illustrates how local scheduler on every processor schedules τ_{π_k} along with T_k^d. Being the highest priority tasks at time instant $t = 0$ on all processors, T_k^d qualifies to execute on all three processors simultaneously. However, once T_1^d starts its execution on π_1, priorities of T_2^d and T_3^d are exchanged with local tasks. Note that, for second job of T_k^d at time instant $t = 3$, T_3^d starts first on π_3 instead of T_1^d on π_1. This dynamic relocation of T_k^d comes from the priority order assigned by local scheduler. For instance, when T_1^d has lower priority than any of the locally partitioned tasks on π_1, it cannot compete for priority among T_2^d and T_3^d present on π_2 and π_3, respectively, and therefore, the order in which T_k^d appears will change.

3.3.3.2 Top-level Scheduler

Top-level scheduler also uses an instance of single-processor optimal EDF scheduling algorithm for migrating sub-set of tasks –i.e., τ_{glob}. Recall that the overall task set τ is considered a priori feasible and the tasks present in τ_{glob} are the tasks that were impossible to be statically partitioned. Thus, the platform resource requirement of τ_{glob} is not more than the under-utilization available in the system. Top-level EDF scheduler visualizes the

Algorithm 3 Local-level scheduler: Online jobs assignment for partitioned tasks present in τ_{π_k}

1: **define** π: processor containing local-level scheduler
2: $Y \leftarrow$ number of processor-groups
3: **for** $i = 1...Y$ **do**
4: $m_i \leftarrow$ number of processors in processor-group i;
5: **for** $k = 1...m_i$ **do**
6: $ReTQ(\tau_{\pi_k}) \leftarrow T_k^d$; {*adds dummy task to local ReTQ of every processor of group* i}
7: **for** every scheduling event **do**
8: **sort** $ReTQ(\tau_{\pi_k})$ w.r.t. EDF priority
9: $T \leftarrow$ highest priority ready task from $ReTQ(\tau_{\pi_k})$;
10: **if** $T = T_k^d$ **then**
11: **for** $k = 1...(m_i - 1)$ **do** {*for all processors other than* π}
12: **if** T_k^d is already running on π_k **then**
13: priority of T_k^d is exchanged;
14: $T \leftarrow$ subsequent priority task from $ReTQ(\tau_{\pi_k})$;
15: *break*;
16: $\pi \leftarrow T$;

fragmented amount of computation power available on different processors, which is accessible in a *sequential* manner. Algorithm 4 illustrates jobs assignment on processors by top-level scheduler. If global ready task queue ($ReTQ(\tau_{glob})$) is not empty then at most Y tasks (here, Y refers to the number of processor-groups in the system) are selected for execution (lines $1-7$) such that each selected task executes over each processor-group. Within each group, the top-level scheduler looks for T_k^d task. If T_k^d is running on any of the processors then selected task from $ReTQ(\tau_{glob})$ for that group is assigned on the processor for at most C_k^d units of time (lines $8-14$). Otherwise, if T_k^d is not running on any of the processors of the selected group then task from $ReTQ(\tau_{glob})$ remains suspended until T_k^d starts running.

Algorithm 4 Top-level scheduler: Online jobs assignment for migrating tasks present in τ_{glob}

1: $Y \leftarrow$ number of processor-groups
2: **sort** $ReTQ(\tau_{glob})$ w.r.t. EDF priority
3: **for** $i = 1...Y$ **do**
4: $m_i \leftarrow$ number of processors in processor-group i;
5: **if** $size(ReTQ(\tau_{glob})) \neq 0$ **then**
6: $T_i \leftarrow$ highest priority ready task among τ_{glob};
7: **for** $k = 1...m_i$ **do**
8: **if** T_k^d is running **then**
9: $\pi_k \leftarrow T_i$; // T_i executes for C_k^d time units on π_k
10: *break*;

Figure 3.5 illustrates that when T_k^d starts executing on a processor, top-level EDF scheduler fills its empty C_k^d with the execution requirement of highest priority task available in τ_{glob} (recall that T_k^d is an empty task). As soon as T_k^d finishes on one processor, top-level scheduler preempts the running tasks from $ReTQ(\tau_{glob})$ and migrates it to the next processor that runs T_k^d within the same processor-group.

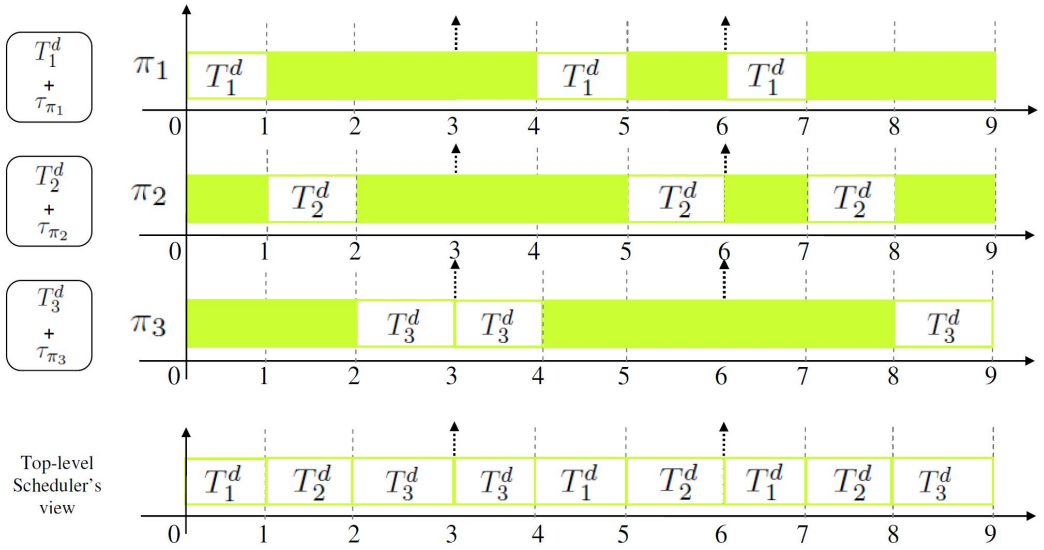


Figure 3.5: View of runtime schedule by top-level and local-level schedulers under 2L-HiSA on an SMP architecture.

3.3.4 Schedulability Analysis

In this section, we provide the reader the schedulability analysis of two-level hierarchical scheduling algorithm. We use *demand bound analysis* for this purpose [63, 77]. Demand bound analysis is a general methodology for schedulability analysis of EDF scheduling algorithm in single-processor systems. Demand bound analysis uses the concept of *demand function (df)*. Demand function computes the maximum amount of time, so-called *processor demand*, consumed by all jobs of a task T_i that have both release times and deadlines within an interval $[t_1, t_2]$. Demand function for a task T_i can be given by equation 3.5.

$$df_i(t_1, t_2) = \sum_{r_{ij} \geq t_1, d_{ij} \leq t_2} C_{ij} \tag{3.5}$$

Similarly, for the entire task set, demand function is simply a sum of individual demand functions of tasks over the same time interval as given by equation 3.6.

$$df(t_1, t_2) = \sum_{i=1}^{n} df_i(t_1, t_2) \tag{3.6}$$

It has been shown in [15, 63] that the EDF-schedulability of arbitrarily-deadline task systems can be tested by the demand function: all tasks are guaranteed to meet deadlines by EDF on single processors, if and only if the condition in equation 3.7 holds for $\forall L > 0$, where $L{=}t_2 - t_1$. On a single-processor system, this is a necessary and sufficient condition for EDF-schedulability.

$$df(t_1, t_2) \leq (t_2 - t_1) \qquad \forall t_1, t_2 \tag{3.7}$$

We divide the schedulability analysis of 2L-HiSA into two parts. In the first part, we analyze the EDF-schedulability of migrating tasks and in second part, we analyze EDF-schedulability of partitioned tasks.

3.3.4.1 Schedulability of migrating tasks

As discussed earlier in section 3.3.2, subset of migrating tasks can not have an aggregate utilization (τ_{glob}) more than the under-utilization available in the system ($U_{sum}(\tau_{glob}) \leq \sum_{k=1}^{m} U'_{\pi_k}$). We have illustrated in figure 3.5 that this under-utilization is proportionately fragmented over different processors of the system and the computation power not more than the equivalent of one processor is freely available within each group. Top-level EDF scheduler, thus, has this fragmented computation power (more than or equal to the cumulated execution requirement of migrating tasks) available in the system to which, it can access in a sequential manner thanks to the explicit occurrence of T_k^d (see figure 3.5). Moreover, T_k^d is the most frequently occurring task on every processor (i.e., it recurs over the smallest period). Thus, migrating tasks always find the portion of processor time reserved for them, which is sufficient w.r.t. their execution requirement. Partitioned tasks, on the other hand, find the remaining non-reserved time units to execute.

3.3.4.2 Schedulability of partitioned tasks in the absence of T_k^d

In a multiprocessor system with fully partitioned task set, the problem of schedulability analysis is reduced to multiple *single*-processor systems. Therefore, it is sufficient to prove that all tasks that are partitioned on a processor $\pi_k (\forall k, 1 \leq k \leq m)$ respect their deadlines. We consider the EDF-schedulability on every processor individually. First, let us consider that only statically partitioned tasks are present on every processor and T_k^d does not exist. We assume that the complementary relation of equation 3.8 and equation 3.9 holds on any processor π_k due to NP-hardness of partitioning problem.

$$U_{\pi_k}(\tau_{\pi_k}) = \sum_{i=1}^{np} \frac{C_i}{P_i} < 1 \quad (\forall i, 1 \leq i \leq np, \forall T_i \in \tau_{part}) \tag{3.8}$$

$$U'_{\pi_k}(\tau_{\pi_k}) = 1 - \sum_{i=1}^{np} \frac{C_i}{P_i} > 0 \quad (\forall i, 1 \leq i \leq np, \forall T_i \in \tau_{part}) \tag{3.9}$$

For synchronous task system, demand function changes values only at discrete time instants corresponding to arrival times and deadlines of a task. Therefore, the demand function needs to be verified only for those values of time intervals that are aligned with deadlines of jobs. Moreover, the worst case demand is found for intervals starting at 0 due to synchronized release instants of all tasks. The hyper-period (i.e., least-common-multiple of task periods) is a safe interval length to analyze demand function for synchronous task sets. Thus, we consider that the worst-case demand interval on every processor π_k is defined from 0 to the hyper-period (let us say H) of partitioned tasks -i.e., $[t_1, t_2]$=$[0, H]$. As long as $U_{\pi_k}(\tau_{\pi_k}) < 1$ –i.e., the aggregate utilization of partitioned tasks is less than the computation power of a single processor, the demand function of all partitioned tasks on processor π_k is strictly less than the amount of time available in the time interval $[0, H]$ as given by equation 3.10. Hence, all partitioned tasks respect the necessary and sufficient schedulability condition of EDF scheduling in the absence of T_k^d on every processor independently. Equation 3.10 also holds for any sub-interval of time $[0, t]$ $(\forall t, 0 < t \leq H)$.

$$\forall t_1, t_2 \qquad df\,(t_1, t_2) \leq (H - 0) \tag{3.10}$$

3.3.4.3 Schedulability of partitioned tasks in the presence of T_k^d

In this section, we consider the EDF-schedulability of partitioned tasks in the presence of T_k^d on every processor individually. In order to be EDF-schedulable, a single-processor system must satisfy the inequality presented by equation 3.11 in the presence of T_k^d.

$$\sum_{T_i \in \tau_{\pi_k}} df_i\left(T_i, H\right) + df(T_k^d, H) \leq H \tag{3.11}$$

The first addend refers to demand function of partitioned tasks and second addend refers to the demand function of T_k^d on processor π_k, respectively. Recalling from section 3.3.2, the size of time slot reserved by T_k^d –i.e., C_k^d, on any processor π_k is in proportion to $U_{\pi_k}^{'}$ available on π_k. Moreover, T_k^d competes for priority at runtime at local scheduler's level, thus, T_k^d is treated as any other partitioned task by the local scheduler.

From equation 3.11, we can deduce that, by design, the demand function of partitioned tasks on processor π_k is always less than or equal to $(H-0) \times U_{\pi_k}$ as shown by equation 3.12. Similarly, from the complimentary relation of equation 3.9, we can deduce that the amount of time allocated to T_k^d is less than or equal to $(H-0) \times U_{\pi_k}^{'}$ as shown by equation 3.13

$$\sum_{T_i \in \tau_{\pi_k}} df_i\left(T_i, H\right) = (H-0) \times U_{\pi_k} \tag{3.12}$$

$$df(T_k^d, H) = (H-0) \times U_{\pi_k}^{'} \tag{3.13}$$

By substitution, the inequalities of equation 3.11 results in equation 3.14.

$$\sum_{T_i \in \tau_{\pi_k}} df_i\left(T_i, H\right) + df(T_k^d, H) \leq H \times \left[\sum_{i=1}^{np} \frac{C_i}{P_i} + 1 - \sum_{i=1}^{np} \frac{C_i}{P_i}\right] \tag{3.14}$$

$$\sum_{T_i \in \tau_{\pi_k}} df_i\left(T_i, H\right) + df(T_k^d, H) \leq H \tag{3.15}$$

Equation 3.15 illustrates that the overall demand function of partitioned tasks together with T_k^d is still less than or equal to the amount of time available in hyper-period (H). Therefore, necessary and sufficient conditions of EDF-schedulability holds at local scheduler-level as well. The bound on schedulable utilization of tasks under the 2L-HiSA scheduling algorithm depends on the following condition.

Condition-I: *Subset τ_{part} shall be partitioned on* m*-processors of the platform in such a way that the under-utilization per group of processors is less than or equal to 1.*

The assignment of tasks to processors is a bin-packing problem, which is considered a strong NP-hard problem [44]. The NP-hardness of partitioning problem can often be a limiting factor for our proposed algorithm. However, the fact that 2L-HiSA makes clusters of identical processors such that, per cluster, the unused fragmented computation power equivalent to at most one processor is available, improves on the schedulable utilization bound of EDF for multiprocessor systems. Clustering of processors instead of considering individual processors, helps in alleviating bin-packing limitations by effectively increasing *bin* sizes in comparison to *item* sizes. With a cluster of processors, it is much easier to obtain the under-utilization per cluster less than or equal to the processing power equivalent to one processor as compared to finding an optimal partitioning of tasks on a single processor. 2L-HiSA is an optimal algorithm for hard real-time tasks if a subset of tasks can be partitioned

such that the under-utilization per cluster of processors remain less than or equal to the processing power equivalent of one processor. General schedulable utilization bound of 2L-HiSA is greater than EDF and less than m. However, the exact bound depends on efficient partitioning.

3.4 Experiments

3.4.1 Setup

In this section, we provide the reader the simulation-based evaluation of the 2L-HiSA scheduling algorithm. Our objective in these experiments is two-folds: 1) to validate whether the analytical improvements claimed on the schedulability bounds of EDF using 2L-HiSA hold in practice and all tasks respect their timing constraints and 2) to analyze performance-related overheads compared to existing optimal scheduling algorithms. We evaluate the performance of 2L-HiSA using STORM (Simulation TOol for Real-time Multiprocessor Scheduling) [108] (see section 2.3 and Appendix A for more information). We consider the same general system model –i.e., task model, processing platform, and power and energy models, as discussed in chapter 2 except that all tasks of target application are not fully migrating. We use synthetic real-time independent and periodic tasks for evaluation. EDF scheduling algorithm is used for both top-level and local-level schedulers.

3.4.2 Functional Evaluation

In this section, we evaluate the functional aspects of 2L-HiSA –i.e., real-time constraints and feasibility aspects. Let us consider a synthetic set of ten real-time periodic and independent tasks ($n = 10$), such that τ=$\{T_5, T_6, T_7, T_8, T_9, T_{10}, T_{11}, T_{12}, T_{13}, T_{14}\}$, to be scheduled on a multiprocessor platform of type SMP composed of four processors ($m = 4$). Table 3.1 presents the quadruplet values of all these tasks. Note that the task names start from T_5 as the initial four tasks names (from T_1 to T_4) are reserved to represent *dummy* tasks T_k^d on processors (from π_1 to π_4), respectively. Task set τ has an aggregate utilization $U_{sum}(\tau) = 4.00$. In the first phase of our algorithm –i.e., the task-partitioning phase (see section 3.3.2), each task is statically assigned to a particular processor by following the bin-packing approach[2]. We obtain $\tau_{part} = \{T_5, T_6, T_7, T_8, T_9, T_{10}, T_{11}, T_{12}\}$ such that τ_{π_1}=$\{T_5, T_6\}$, τ_{π_2}=$\{T_7, T_8\}$, τ_{π_3}=$\{T_9, T_{10}\}$, τ_{π_4}=$\{T_{11}, T_{12}\}$, and τ_{glob}=$\{T_{13}, T_{14}\}$.

Now, we simulate τ under the 2L-HiSA scheduling algorithm using STORM simulator. In the first stage, only τ_{part} is executed on Π. Figure 3.6 illustrates that idle time intervals appear on every processor due to under-utilization. We calculate this under-utilization using equation 3.9 as following.

$$U_{\pi_1}^{'} = 1 - 0.70 = 0.30$$

$$U_{\pi_2}^{'} = 1 - 0.70 = 0.30$$

$$U_{\pi_3}^{'} = 1 - 0.80 = 0.20$$

$$U_{\pi_4}^{'} = 1 - 0.80 = 0.20$$

[2]Tasks are partitioned manually to processors. This may not be the best possible partitioning solution for given task set, but it is good enough to illustrate the functioning of the 2L-HiSA algorithm. However, efficient task partitioning approaches can be used in this phase.

Table 3.1: Real-time periodic task set τ

Task Name	r_i	C_i	d_i	P_i
T_5	0	6	20	20
T_6	0	6	15	15
T_7	0	13	40	40
T_8	0	15	40	40
T_9	0	6	30	30
T_{10}	0	12	20	20
T_{11}	0	8	20	20
T_{12}	0	10	25	25
T_{13}	0	6	10	10
T_{14}	0	8	20	20

Once the under-utilization per processor is known, we perform processor-grouping –i.e., the second phase of our algorithm, such that Condition-I is satisfied –i.e., $\sum_{k=1}^{m} U'_{\pi_k} \leq 1$ (see section 3.3.4). Since the cumulative under-utilization of all processors is exactly equal to one in this case ($\sum_{k=1}^{m} U'_{\pi_k} = 1$), therefore, all processors are grouped together in a single group. After task-partitioning and processor-grouping phases are complete, we can add dummy task T_k^d on every processor in proportion to the available under-utilization. The parameters of T_k^d such as, its period P_k^d and worst-case execution time C_k^d are calculated using equation 3.3 and 3.4, respectively. From table 3.1, it is straightforward to obtain the smallest period of all tasks, that is:

$$P_k^d = \min_{i=1}^{n} \{P_i\} = 10$$

For the known value of period, we can calculate the worst-case execution time of T_k^d on every processor π_k w.r.t. U'_{π_k} available on that processor over P_k^d as following.

$$C_1^d = P_1^d \times U'_{\pi_1} = 10 \times 0.30 = 3$$

$$C_2^d = P_2^d \times U'_{\pi_2} = 10 \times 0.30 = 3$$

$$C_3^d = P_3^d \times U'_{\pi_3} = 10 \times 0.20 = 2$$

$$C_4^d = P_4^d \times U'_{\pi_4} = 10 \times 0.20 = 2$$

Table 3.2: Parameters of dummy tasks (T_k^d) on each processor

Task Name	r_i	C_i	d_i	P_i
T_1^d	0	3	10	10
T_2^d	0	3	10	10
T_3^d	0	2	10	10
T_4^d	0	2	10	10

All parameters of dummy tasks T_k^d are summarized in table 3.2. Figure 3.7 illustrates simulation traces generated by local-level EDF scheduler on each processor in the presence

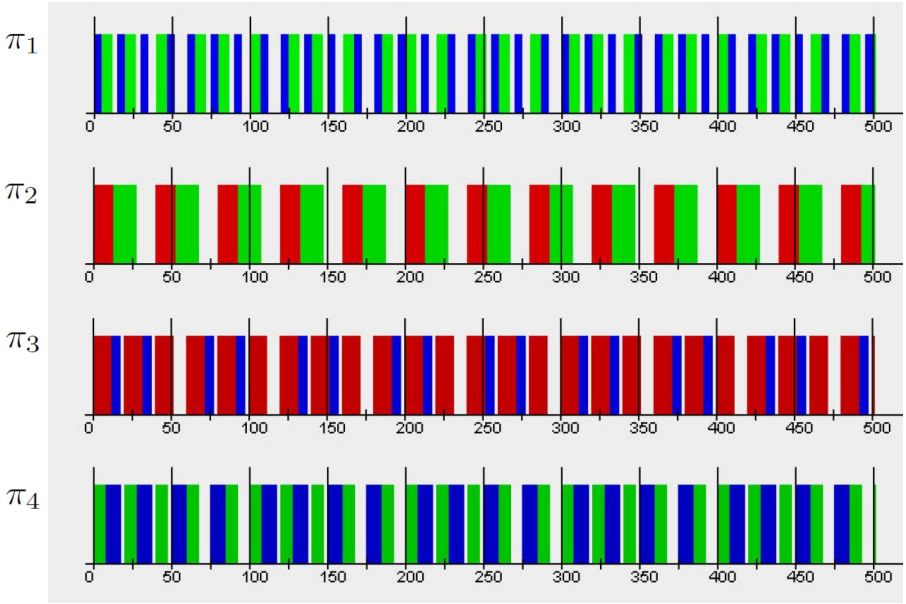


Figure 3.6: Simulation traces of partitioned tasks under EDF local scheduler on each processor.

of T_k^d (note that simulator outputs the task names as PTASK *taskname*). It can be noticed in this figure that T_k^d tasks appear sequentially on processors. In figure 3.7, task T_1^d (or *PTASK* T_1) appears first on processor π_1 for exactly 3 time units. Since, T_1^d starts its execution first, therefore, T_2^d, T_3^d, and T_4^d on π_2, π_3, and π_4, respectively, exchange their priorities to let the partitioned tasks execute. Once T_1^d finishes its execution, T_2^d starts executing on π_2 for its corresponding worst-case execution time (i.e., $C_2^d = 3$). This process repeats itself for all dummy tasks within each period P_k^d. As mentioned in section 3.3.2, note that T_k^d has neither a specific order of occurrence nor it is fixed a priori on processors. Rather it can dynamically relocate itself within P_k^d. Every time T_k^d is released, first, it has to compete for the priority among locally partitioned tasks (thanks to the choice of smallest period, T_k^d often has highest priority among locally partitioned tasks) and then compete for priority among T_k^d present on other processors within a group. Failure to obtain highest priority at any of these two levels cause a priority inversion for concerned task itself and corresponding processor can execute locally partitioned ready tasks (if any). However, once any of the T_k^d tasks start executing, no other T_k^d tasks can execute in parallel. The priority inversion for T_k^d on the same processor can be performed until the laxity of T_k^d becomes zero. For instance, T_4^d in figure 3.7 starts executing at time instant $t = 8$ at which, its laxity becomes zero –i.e., $L_4^d = 10 - (8 + 2) = 0$. After instant $t = 8$, it was no more possible to invert the priority of T_4^d without a deadline miss.

Finally, figure 3.8 illustrates a complete simulation trace of τ under two-level hierarchical scheduling algorithm along with T_k^d. Figure 3.8 illustrates that as long as any of the T_k^d task is executing on any of the processors in platform, top-level scheduler can manage to *fill* its C_k^d with the execution time of highest priority migrating task. Moreover, top-level scheduler can preempt and migrate the migrating task(s) to other processor(s) whenever any of the T_k^d task finishes on a processor. In figure 3.8, blue rectangular boxes on the time

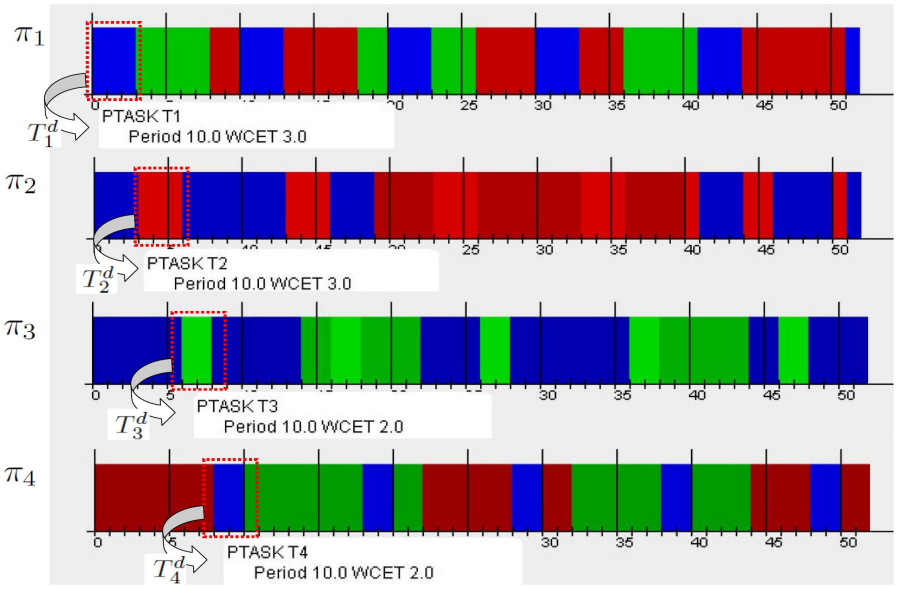


Figure 3.7: Simulation traces of partitioned tasks in the presence of T_k^d under EDF local scheduler on each processor.

scale represent migrating task T_{13} and red rectangular boxes represent migrating task T_{14}.

We simulate the same task set presented in table 3.1 using global EDF scheduling algorithm in order to illustrate the sub-optimality of EDF. Figure 3.9 illustrates the execution traces for τ. It can be noticed that despite 100% workload ($U_{sum}(\tau) = 4.00$), some processors still remain momentarily idle. In figure 3.9, these idle time intervals are highlighted with dotted line boxes. Idle time intervals appear due to the priority mechanism of global EDF algorithm. Figure 3.10(a) and 3.10(b) illustrate that, due to the priority mechanism of EDF, some of the lower priority tasks, for instance, T_{12} and T_{13} in this case, miss their deadlines[3]. This illustration validates theoretically known sub-optimality of global EDF scheduing algorithm.

3.4.3 Energy-efficiency of 2L-HiSA

In this section, we provide the reader the possibilities of applying online energy-aware scheduling techniques such as; dynamic power management and dynamic voltage and frequency scaling techniques in conjunction with 2L-HiSA.

In section 3.3.4, we have provided the worst-case schedulability analysis of 2L-HiSA. This is a rather conservative analysis because during execution, real-time tasks often exhibit large variations in their actual execution time. Tasks often finish earlier than their estimated worst-case execution time and generate dynamic slack [39]. In this section, we implement a simple DVFS technique under the control of 2L-HiSA in which, whenever a precedent task generates dynamic slack time, the entire amount of slack is used to slowdown the execution of immediate priority ready task on the same processor –i.e., all slack is consumed by

[3]Jobs for which deadline miss occurs is highlighted with oval-shaped red box beneath the simulation trace of each task.

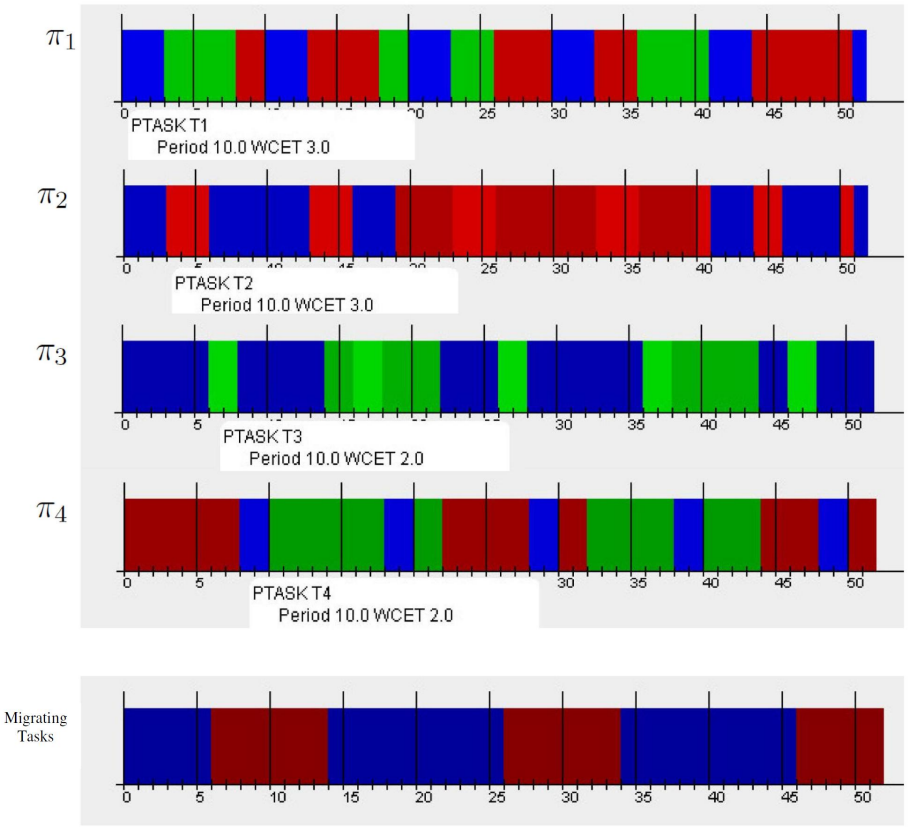


Figure 3.8: Simulation traces of migrating and partitioned tasks together under EDF local- and top-level schedulers.

the next appropriate priority task. We have applied DVFS technique under the control of local-level scheduler on each processor. Only statically partitioned tasks benefit from this slack reclamation –i.e., whenever a partitioned task generates dynamic slack, only the next partitioned ready task can consume it. Otherwise, the slack time is considered as lost. However, it would also be possible to *share* slack between top-level and local-level schedulers on each processor –i.e., whenever a partitioned task produces dynamic slack by finishing early, the size of T_k^d's time window (i.e., C_k^d) on that particular processor can be dynamically enlarged to benefit from the workload variations. Slack sharing between hierarchy of schedulers is not implemented.

We have simulated the task set presented in section 3.4.2 under global EDF scheduler, under 2L-HiSA without DVFS technique, and under 2L-HiSA with DVFS technique. We have kept the number of tasks and their aggregate utilization constant, however, the actual execution time or bcet/wcet ratio of tasks is varied between 50% and 100% of their worst-case execution time (C_i). The variation is auto-generated such that the actual execution time (AET) has a uniform probability distribution function as suggested in [11]. Energy consumption is estimated for processors only. Simulation results depict that, in best-case –i.e., for bcet/wcet ratio = 50%, the energy savings can reach upto 42.6% under 2L-HiSA with DVFS technique as compared to non-optimized EDF schedule. Furthermore, the energy gains under 2L-HiSA with DVFS technique as compared to 2L-HiSA without DVFS

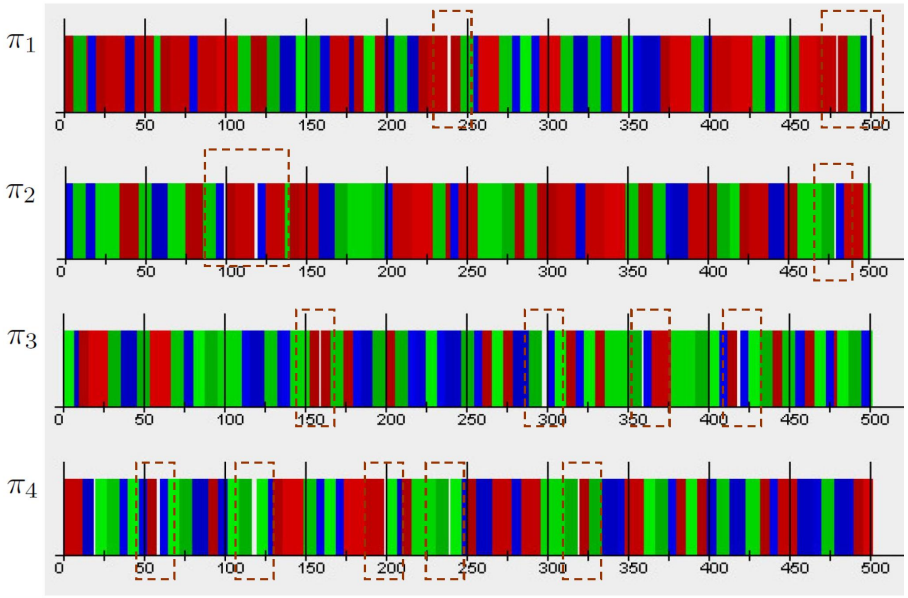


Figure 3.9: Simulation traces of EDF global scheduling of task set τ on four processors.

technique are estimated upto 44.7%. Simulation results show that applying such simple DVFS techniques does not yield a very significant difference between 2L-HiSA with and without DVFS technique. However, energy gains are remarkable as compared to non-optimized EDF schedule.

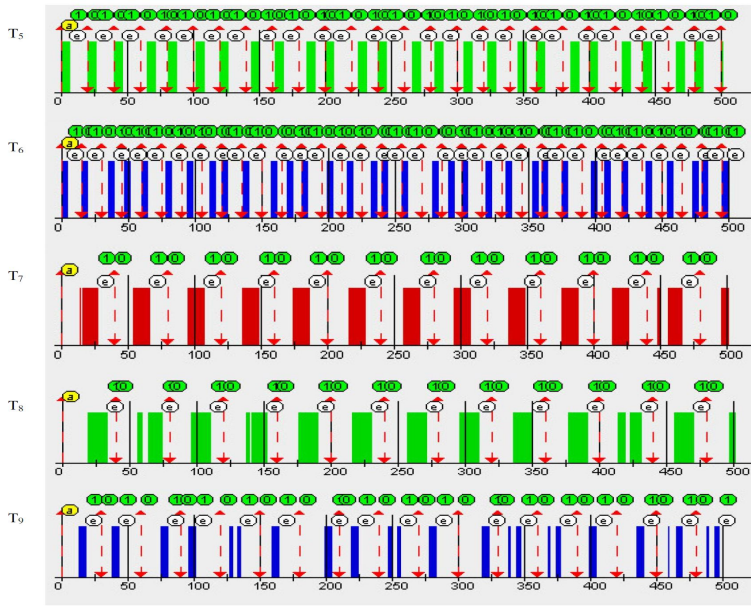
We have three intuitive remarks on these results. Firstly, these results illustrate that it is possible to integrate other energy-aware scheduling techniques with 2L-HiSA without loss of schedulability of tasks. Secondly, even for processors only, 2L-HiSA gives significant energy-efficiency. Thirdly, restricted-migration scheduling strategies naturally favor memory subsystems, especially L1 caches, for energy-efficiency due to reduced task migration and more or less constant cache-contents.

3.4.4 Performance Evaluation

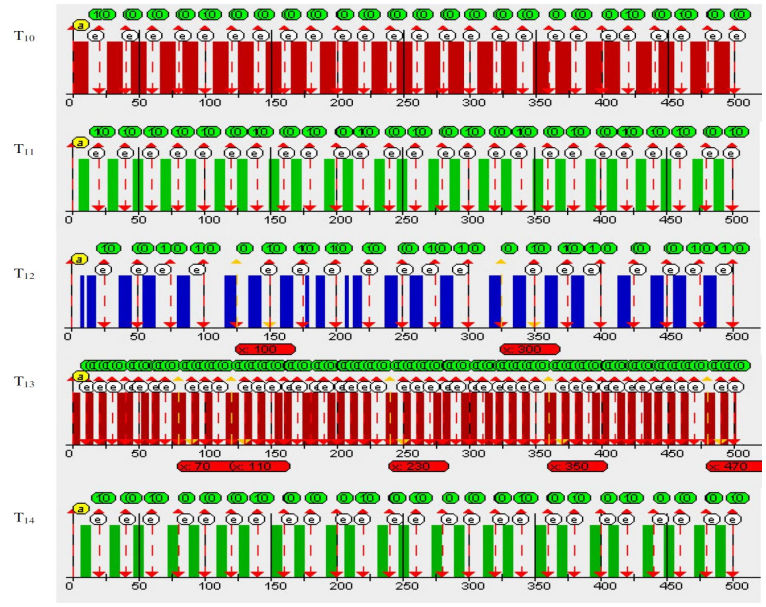
In this section, we provide analysis of the performance of 2L-HiSA as compared to already existing global optimal scheduling algorithms such as PFair [14], LLREF [28], and ASEDZL [77] algorithms.

PFair and its heuristic algorithms are based on the concept of fluid scheduling mechanism in which, they select tasks to execute at each time instant. Doing so invokes the scheduler at every time instant, which introduces a lot of overhead in terms of increased release instants (r_i), task preemptions, and migrations. PFair is often criticized for its scheduling-related complexity. Unlike PFair, LLREF algorithm[4] is not based on time quanta but it increases preemptions of tasks to a great extent. LLREF schedules all ready tasks between

[4]Although, we compare performance of LLREF algorithm with other algorithms analytically, we are unable to provide comparative analysis based on simulations due to our development limitations.



(a) Simulation traces for $T_5 - T_9$.



(b) Simulation traces for $T_{10} - T_{14}$.

Figure 3.10: Simulation traces of individual tasks under global EDF scheduler

any two release instants. Since all tasks are *active* at all time instants, therefore, context-switching overhead and cache-related preemption delay is significantly large for LLREF. ASEDZL algorithm, contrary to PFair, is not based on time quanta. Execution requirement and time periods of tasks can have any arbitrary value under ASEDZL algorithm. It improves on LLREF algorithm by scheduling minimum number of tasks between any

two release instants. However, it still incurs higher number of scheduling events and preemptions than EDF scheduler. 2L-HiSA scheduling algorithm uses multiple instances of single-processor optimal EDF algorithm to schedule tasks both at top-level and local-level schedulers. Since, EDF invokes the scheduler only at job boundaries, therefore, the overhead in terms of release instants and number of preemptions is much less than the techniques discussed earlier. Furthermore, 2L-HiSA has reduced overhead of L1 cache memories due to the limited number of context-switches. Most of the tasks are partitioned under this algorithm, which limits the number of task migrations (only migrating or global sub-set of tasks migrate). Thus, the caches are mostly occupied by partitioned tasks, which helps in reducing the recovery time that a task may suffer from cache-miss and eventually improve performance.

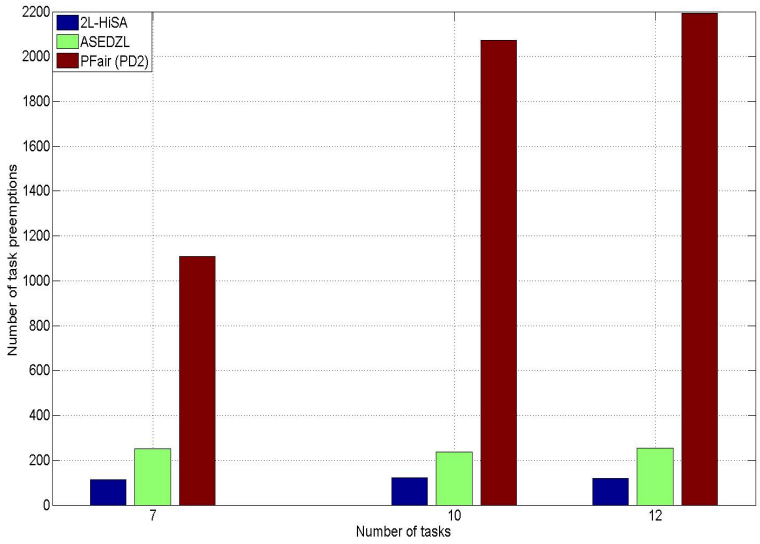


Figure 3.11: Number of task preemptions under 2L-HiSA, PFair (PD^2), and ASEDZL algorithms.

We compare the number of task preemptions and task migrations under 2L-HiSA, PD^2 PFair algorithm [6], and ASEDZL [77] algorithms for the task set presented in table 3.1 over a simulation time equal to one hyper-period –i.e., 600 time units. Figure 3.11 illustrates that the number of preemptions under PD^2 PFair algorithm for various number of tasks is the highest. We have estimated an average difference of 15-fold between preemptions under PD^2 PFair and ASEDZL and an average difference of 18-fold between PD^2 PFair and 2L-HiSA. An average difference in the number of preemption between ASEDZL and 2L-HiSA has been estimated up to 1.3-fold. Note that these results take into account the preemptions of tasks under every local-level scheduler as well as top-level scheduler while using 2L-HiSA. Similarly, figure 3.12 illustrates the number of task migrations for various number of tasks. Still, migration of tasks under PD^2 PFair algorithm is relatively very high. We have estimated an average difference of 4-fold between task migration under PD^2 PFair and ASEDZL and an average difference of 10-fold between PD^2 PFair and 2L-HiSA. An average difference in the number of task migration between ASEDZL and 2L-HiSA has

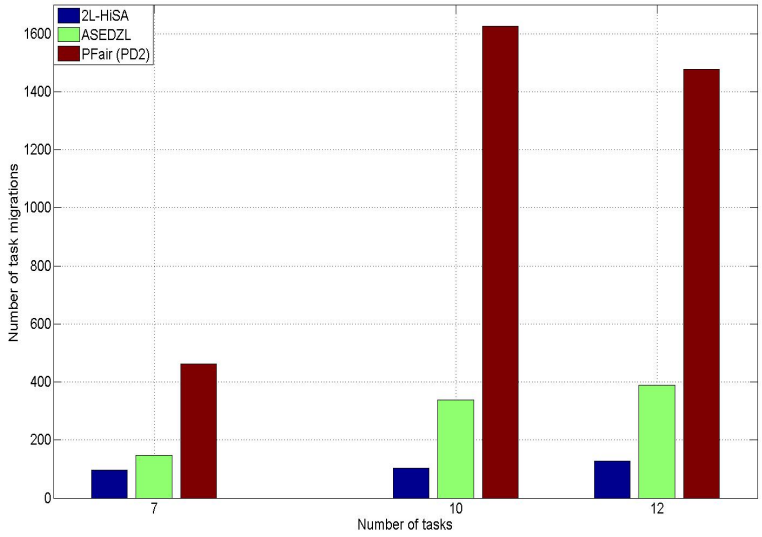


Figure 3.12: Number of task migrations under 2L-HiSA, PFair (PD^2), and ASEDZL algorithms.

been estimated up to 2.6-fold. These results show that using the 2L-HiSA algorithm can be benevolent from performance point of view.

3.5 Concluding Remarks

In this chapter, we present a multiprocessor scheduling algorithm, called two-level hierarchical scheduling algorithm (2L-HiSA), which falls in the category of restricted migration scheduling. The EDF scheduling algorithm has the least runtime complexity among job-level fixed-priority algorithms for scheduling tasks on multiprocessor architecture. However, EDF suffers from sub-optimality in multiprocessor systems. 2L-HiSA addresses the sub-optimality of EDF as global scheduling algorithm and divides the problem into a two-level hierarchy of schedulers. We ensure that the basic intrinsic properties of optimal single-processor EDF scheduling algorithm appear both at local-level as well as at top-level scheduler. This algorithm works in two phases: 1) A task-partitioning phase in which, each task from application task set is assigned to a specific processor by following simple bin-packing approach. If a task can not be partitioned on any processor in the platform, it qualifies as migrating task. 2) A processor-grouping phase in which, processors are clustered together such that, per cluster, the unused fragmented computation power equivalent to at most one processor is available.

2L-HiSA improves on the schedulability bound of global EDF for multiprocessor systems and it is optimal for independent and periodic hard real-time tasks if a subset of tasks can be partitioned such that the under-utilization per cluster of processors remain less than or equal to the computation power equivalent to at most one processor. The NP-hardness of partitioning problem, however, can often be a limiting factor. By clustering of processors instead of considering individual processors, 2L-HiSA alleviates bin-packing

limitations by effectively increasing *bin* sizes in comparison to *item* sizes. With a cluster of processors, it is much easier to obtain the under-utilization per cluster less than or equal to the computation power equivalent to one processor. This chapter provides simulation results to support our proposition. We have illustrated that power- and energy-efficient strategies like DVFS and/or DPM can be used in conjunction with 2L-HiSA to improve energy savings. Furthermore, we have illustrated that the task preemption- and migration-related overhead is significantly less while using 2L-HiSA as scheduling algorithm.

Chapter 4

Assertive Dynamic Power Management Technique

Contents

4.1 Dynamic Power Management

Dynamic Power Management(DPM) techniques achieve energy conservation in embedded computing systems by actively changing the power consumption profile of the system by selectively putting its components into power-efficient states sufficient to meeting functionality requirements [57]. These techniques, when applied, exploit the inherently present idle time intervals (if any) in the real-time schedule of target application over a specific target architecture at runtime. The idle time intervals can exist due to the variations in actual workload at runtime or they can be statically present due to under-utilization of target platform. Optimality of DPM techniques depends on the workload statistics. Based on these statistics, the effectiveness of different DPM techniques vary.

In this chapter, we present a DPM technique for multiprocessor real-time systems, called the *Assertive Dynamic Power Management (AsDPM)* technique. This technique is intended for multiprocessor real-time platforms of type SMP and it works under the

control of global EDF and LLF scheduling algorithms[1]. Our proposed technique is called *assertive* DPM technique because of its aggressiveness in *extracting* the idle time intervals (if present) from the application's runtime schedule, which is not the case in conventional DPM techniques. The working principle presented in section 4.3 further elaborates this aspect. Apart from being a DPM technique for processors, AsDPM can be useful in optimizing energy consumption at memory subsystem level as well. Moreover, the AsDPM technique can be used, based on the workload characteristics, for statically (offline) optimizing the number of processors and their corresponding operating frequency and voltage level in a multiprocessor system. As a future perspective, we discuss how heat dissipation can be regulated while using this technique. Rest of this chapter is arranged as follows. In section 4.2, we review state-of-the-art on DPM techniques. AsDPM is presented in detail in section 4.3. Use of AsDPM for static optimizations in discussed in section 4.4. Experimental evaluation is presented in section 4.5. Section 4.6 provides a brief discussion on ongoing and future perspectives of the AsDPM technique.

4.2 Related Work

In two different surveys of DPM techniques presented in [16] and [102], authors classify DPM techniques into two main categories: *predictive* schemes and *stochastic* schemes. Predictive schemes attempt to predict the timing of future input events to the system and schedule shutdown (usually to a single power-efficient state) based on these predictions. Usually, not much is known regarding future input events in an online schedule and DPM decisions have to be taken based on these so called uncertain predictions. The rationale in all predictive schemes is to exploit the correlation between past history of the workload and its near future in order to make reliable predictions about future events [53, 102]. The most common predictive DPM policy is the *fixed timeout* DPM policy, which uses the elapsed idle time as observed event to predict the total duration of the current idle period. Such policies have obvious advantage of being general and their safety can be improved simply by increasing the timeout values. The obvious disadvantage of these techniques is that they trade-off efficiency for safety: large timeouts cause a large number of under-predictions, that represent missed opportunity of saving power, and a sizable amount of power is wasted waiting for the timeout to expire. The *predictive shutdown* policies improve upon timeout-based schemes by taking decisions as soon as a new idle period starts, based on the observation of past idle and busy periods. One of the early research work on such techniques was presented in [107] in which, authors have proposed a predictive DPM technique for event-driven applications. The basic idea in [107] is to predict the length of idle periods and shutdown the system when the predicted idle period is long enough to amortize the cost (in latency and power) of shutting down and later reactivating the system. A shortcoming of predictive shutdown approach is that it is based on offline analysis of usage traces, hence it is not suitable for non-stationary workload whose statistical properties are not known a priori. This shortcoming is addressed by Hwang and Wu in [53]. They proposed *online adaptive* methods that predict the duration of an idle period with an exponentially weighted average of previous idle periods. Predictive DPM techniques improve on performance penalty as well, that is always paid on wakeup, by performing *predictive wakeup* when the predicted idle time expires, even if no new task service requests have arrived. This choice may increase power

[1] Although we have implemented the AsDPM technique with both the EDF and the LLF scheduling algorithms, rest of this chapter will consider only EDF being the controlling scheduling algorithm.

dissipation if idle time has been under-predicted, but decreases the delay for servicing the first incoming request after an idle period.

In some relatively recent work, researches have addressed the problem of inaccurate *predictions* in DPM strategies in the real-time context. In real-time systems, predicting the next time instant when a device will be needed is crucial not only because it is the key for effective energy management, but also because inaccurate predictions may hurt the feasibility of the system, in view of the non-trivial activation/de-activation delays. Hence, the real- time DPM has a number of unique characteristics. Authors in [112, 109] have proposed heuristic-based DPM schemes that can be used with both EDF and RMS scheduling polices. However, the schemes consider only non-preemptive real-time task execution. In another work presented in [110], the same group of authors present an offline scheme, called Maximum Device Overlap (MDO), for preemptive task scheduling. The MDO algorithm uses a real-time scheduling algorithm, e.g., EDF, to generate a feasible real-time job schedule and then iteratively swaps job segments to reduce energy consumption in device power state transitions. After the heuristic-based job schedule is generated, the device schedule is extracted. That is, device power state transition actions and times are recorded prior to runtime and used at runtime. However, a problem with MDO is that the schedule is generated with jobs' WCET. Even without resource blocking, the actual job executions can be very different from the pre-generated job schedule. A fixed device schedule cannot effectively adapt to changes in the operating environment, such as if dynamic task join or jobs complete early. Moreover, MDO involves very high time complexity (proportional to the square of the hyper-period) and it cannot be successfully adapted to dynamic/online settings where job release and execution times can vary considerably. *Adaptive predictive* DPM techniques have also been proposed to deal with non-stationary workloads [102].

In another recent research work presented in [36], authors have proposed their solution to the online real- time DPM problem, which is based on the observation that creating long device sleep intervals is the key for effective power management. Authors propose to *explicitly* and *periodically* enforce such intervals for each system's device at run time. These intervals, called the *device forbidden regions* (DFRs), enable the system to put these devices to sleep states. Authors propose to determine parameters, such as duration and separation time of DFRs, through *static* analysis. Authors also ensure that none of the statically determined DFRs should have its duration shorter than the device's break-even times. This approach, however, can only exploit the inherently present idle time intervals in the *offline* schedule to determine DFRs. Moreover, during a forbidden region, none of the tasks using the related device can be dispatched; hence, the duration and period of DFRs must be carefully selected to preserve temporal correctness. This limits the possibility of fully exploiting the offline slack time. The proposed approach in [36] is unable to exploit the *online* slack time (online slack time can be generated due to the early completion of tasks) that can be used for further enlargement of device sleep intervals at runtime. The same authors have extended their DFR-based DPM solution to a unified framework in [35]. The proposed framework, called DFR-EDF, assumes a general system- level energy model and includes both static and dynamic (online) components. The static part is based on the extension of their work in [36] for preemptive EDF scheduling. The online component integrates the predictive DPM techniques and offers a generalized slack reclaiming mechanism that can be used by DVFS and DPM simultaneously. This framework is intended for system-wide energy-efficiency. In this work, authors claim to develop a unified energy management framework for deadline-driven periodic real-time applications by exploiting the interplay between DVFS and DPM with both static and dynamic (runtime) solution components. This interplay of DPM and DVFS techniques, however, is not completely

online and adaptive. DPM is mainly used for static optimizations using device forbidden regions.

Stochastic DPM techniques, rather than eliminating uncertainty by prediction, formulate policy optimization as an optimization problem under uncertainty [18, 57, 84]. More specifically, power management optimization has been studied within the framework of controlled Markov processes. In this case, it is assumed that the system and the workload can be modeled as Markov chains [16, 55]. Under this assumption, it is possible to: model the uncertainty in system power consumption and response (transition) times; model complex systems with many power states, buffers, queues, etc.; compute power management policies that are globally optimum; explore trade-offs between power and performance in a controlled fashion. Stochastic optimum control is a well-researched category of DPM techniques [29, 90, 100]. These techniques have several advantages over predictive techniques. First, it captures the global view of the system, thus allowing the designer to search for a global optimum that possibly exploits multiple inactive states of multiple interacting resources. Second, it enables the exact solution (in polynomial time) of the performance-constrained power optimization problem. Third, it exploits the strength and optimality of randomized policies [29, 83]. However, stochastic optimization techniques assume complete a priori knowledge of the system and its workload statistics which can be a limiting factor in some cases. Even though it is generally possible to construct a model for the system once for all, the workload is generally much harder to characterize in advance. Furthermore, workloads are often non-stationary.

DPM techniques can be applied at several different levels in a system such as at processor-level, at peripheries, or a combination of both. Previously, significant amount of research has focused on applying (both predictive and stochastic) DPM techniques on either processors only [18, 29, 55, 57, 100] or on peripheral devices only [26, 113]. However, system-wide energy conservation has received little attention. Some recent research work has focused on applying DPM techniques together on processors as well as other system components such as peripheral or I/O devices, memory subsystems, flash drives, and wireless network interfaces in order to achieve system-wide energy savings. Authors in [27] suggest that these devices are pervasive in modern embedded systems which consume considerable amount of energy. Authors in [27] propose an online system-wide energy-efficient scheduling algorithm called SYS-EDF, which integrates dynamic power management for I/O devices and dynamic voltage scaling for the processor. SYS-EDF algorithm supports periodic task sets with non-preemptive shared resources. This algorithm takes advantage of slowed-down execution of tasks on processors by applying DVS on processors and consequently, elongated idle periods on unused I/O devices.

The AsDPM technique considers mainly the processors for power and energy consumption optimization. It is not based on predictive mechanisms that predict either the arrival time or the length of an already arrived idle interval. Rather, it works on the principle of admission control for ready tasks by delaying the execution of ready tasks as much as possible, thereby controlling the maximum number of active/running processors in the system at any time instant. Tasks are delayed based on their remaining runtime laxity in a deterministic way. Section 4.3 further elaborates this concept. Although AsDPM is also based on on the observation that creating long device sleep intervals is the key for effective power management, unlike presented in [36], AsDPM is not based on device forbidden regions. Moreover, it can be used to exploit statically present idle time intervals as well as dynamic slack time generated by the early completion of tasks. Unlike [36] and [35], AsDPM does not need to be coupled with other DVFS or predictive DPM techniques in order to exploit

dynamic slack. Also, unlike [110], AsDPM does not extract the device schedule -i.e., device power state transition actions and times are not recorded prior to runtime.

4.3 Assertive Dynamic Power Management Technique

In the following, we define some notations related to AsDPM technique, which are used in this chapter.

Notations:

Scheduler's Task Queues: In order to simplify later discussion, we define certain task queues. Their role is specified as following:

1. **Tasks Queue (TQ):** Contains all tasks which are neither running nor ready/deferred at any point in time.
2. **Released Tasks Queue (ReTQ):** Contains tasks that are released but not executing currently on any processor due to their priority level. When released, a task is immediately put in this queue.
3. **Running Tasks Queue (RuTQ):** Contains tasks that are released and currently running on some processors. When a task T_i is running on a processor π_j, it is represented as $\pi_j[T_i]$ $(\forall i, j; 1 \leq i \leq n, 1 \leq j \leq m)$.
4. **Deferred Tasks Queue (DeTQ):** Contains tasks that are released but deferred from execution. A released task that is not the highest priority task but has its priority high enough to execute on an m-processor platform (i.e., it is among the m highest priority tasks) can be deferred from execution under AsDPM at any scheduling event. Such a task is said to be deferred task. When released, this task is put in ReTQ and upon analyzing its priority, this task is either put in RuTQ (if the highest priority task) or in DeTQ. When a task is deferred, it is assigned a *virtual affinity* to one of the processors in the system and represented as $T_i[\pi_j]$ $(\forall i, j; 1 \leq i \leq n, 1 \leq j \leq m)$. Criterion of assigning this virtual affinity and its advantage is discussed later in section 4.3.1. All tasks present in DeTQ are referred as subset τ^{deff}.

Anticipative Laxity (l_i): We define the anticipative laxity of a task within the context of AsDPM only as follows.

Definition: Anticipative laxity of a task's job is the measure of its urgency *to execute relative to its deadline in the presence of all higher-priority released job(s) running and deferred on a particular processor.*

It differs from absolute laxity[2] is the sense that, in a multiprocessor system, a job having zero or negative anticipative laxity does not imply that it will eventually miss its deadline. Rather, it is used to project future execution of low priority jobs in the presence of higher priority jobs by AsDPM. Anticipative laxity is further elaborated in section 4.3.1.

The proportion of processors' time that an entire real-time task set τ will require in worst-case is given by $U_{sum}(\tau) \stackrel{def}{=} \sum_{i=1}^{n} u_i$ (section 2.1.1.2). For a specific architecture platform composed of m processors, the worst-case under-utilization caused by such task

[2]See section 2.1.1 for the definition of absolute laxity L_i of task.

set is $m - U_{sum}(\tau)$. This under-utilization appears at runtime in the form of random and variable length idle time intervals on different processors for non-stationary workload. A conventional DPM technique, whether predictive or stochastic, can exploit these idle intervals *only* once they occur on a processor –i.e., once an idle interval is detected. Upon detecting idle time intervals, these techniques decide whether to transition target processor(s) to power-efficient state. AsDPM, on the other hand, differs from the existing techniques in the way it exploits idle time intervals. It aggressively extracts all idle time intervals from some processors (let us say k processors) and clusters them on some other processors (let us say m-k processors) to elongate the duration of idle time on $m - k$ processors. Transitioning these $m - k$ processors to suitable power-efficient state then becomes a matter of comparing idle interval's length against the break-even time (BET) of target processor. It is worth mentioning here that AsDPM is an admission control technique for real-time tasks that makes a feasible task set energy-efficient by deciding when exactly a ready task shall execute. Without this admission control, all ready tasks are executed as soon as there are enough computing resources (processors) available in the system, leading to poor possibilities of putting some system components in power-efficient states. AsDPM works only in conjunction with multiprocessor global scheduling algorithms. Moreover, it does not intervene in the priority-related decision making of the scheduling algorithm. That is, the priority order of all tasks, whether deferred or executing, is preserved while using AsDPM in conjunction with global scheduling algorithms. However, a released lower priority job may be deferred/delayed from execution unless its urgency level becomes critical. In order to control the admission of tasks into RuTQ, AsDPM performs a test, called *Laxity Bottom Test (LBT)*, on all released jobs of tasks (except the highest priority job) at every scheduling event.

4.3.1 Laxity Bottom Test (LBT)

Laxity bottom test is performed at every scheduling event for all tasks (except the highest priority job) present in ReTQ. It is based on the anticipative laxity l_i of tasks. Every time LBT is performed, at first, it is assumed that at most one processor is running in the system[3] (i.e., present in active state) at current scheduling event that invoked the scheduler. This assumption allows to assign (and/or defer) maximum tasks on one processor as long as possible. The highest priority ready task from ReTQ is assigned to processor directly for execution and for all remaining ready jobs in ReTQ, LBT is performed. LBT works as follows. If the anticipative laxity of a ready job results in a negative value (< 0) then the test is said to be *failed* for that job and the job would miss its deadline if no further computing resources are provided (i.e., processor must be activated). On the other hand, if LBT passes for a task –i.e., the anticipative laxity results in a positive value (≥ 0) then the job in question can still meet its deadline in future, if it would be *deferred* from execution at current scheduling event (until the next event). LBT ensures deadlines for all released jobs between successive scheduling events. In the following, we describe how anticipative laxity is calculated.

Recall the definition of anticipative laxity from section 4.3, which states that *all* higher priority released and deferred tasks should be considered while calculating the value of anticipative laxity l_i of a particular low priority task T_i on a target processor π_j. In a multiprocessor system, where tasks may execute in parallel, multiple higher priority tasks

[3]There can be more than one processors running in the system. This is just an assumption to start the LBT test. Once the LBT is complete for all tasks, resulting number of required running processors can be the same as previously available or it may be different.

(than T_i) might be simultaneously running on different processors and some higher priority tasks might be already present in τ^{deff}. Moreover, some or all higher priority tasks might have already completed a portion of their execution requirement (C), therefore, only the remaining execution requirement (C^{rem}) of higher priority tasks shall be considered for calculating l_i. Since all higher priority tasks are not supposed to run on π_j sequentially (rather, there will be at most $(m-1)$ higher priority tasks running in parallel), therefore, considering the C^{rem} of all higher priority tasks would result in a very pessimistic value of l_i on π_j. To avoid this pessimism in calculation, anticipative laxity l_i of task T_i is calculated on processor π_j in the presence of *only* those higher priority tasks which are either currently running on π_j or are deferred earlier with an associativity with π_j processor –i.e., $T_k[\pi_j]$ $(\forall k; i < k)$. Here, T_k refers to any task having higher priority than T_i and currently running or deferred on processor π_j. Equation 4.1 illustrates how l_i for task T_i is calculated on processor π_j in the presence of higher priority tasks.

$$l_i = d_i - \left(t + C_j^{rem} + C_i^{rem} + \sum_{T_k[\pi_j] \in \tau^{deff}} C_k^{rem} \right) \tag{4.1}$$

In equation 4.1, t refers to the time instant at which the scheduling event has occurred, C_i^{rem} refers to the remaining execution time of task T_i for which l_i is calculated, C_j^{rem} refers to the remaining execution time of task which is currently running on processor π_j, and C_k^{rem} refers to the remaining execution time of all higher priority deferred tasks present in τ^{deff} having an affinity with processor π_j. When a task is deferred –i.e., its LBT is passed on a processor, the task is assigned a *affinity* to that processor. This affinity is valid between two successive scheduling events only as the LBT is performed again at next scheduling event. The purpose of assigning this affinity is to make the calculation of anticipative laxity non-pessimistic by avoiding the reconsideration of those higher priority tasks that are already being deferred to some other processors. Following example elaborates the working of LBT further.

Example–1: Let us consider a task set composed of three periodic, independent, and synchronous tasks such that τ=$\{T_1, T_2, T_3\}$. The quadruplet (r_i,C_i,d_i,P_i) values for T_1, T_2, and T_3 are $\{0, 6, 12, 12\}$, $\{0, 8, 14, 14\}$, and $\{0, 5, 20, 20\}$, respectively. Task set τ is scheduled under preemtive EDF scheduling policy –i.e., smaller the deadline, greater the priority. In figure 4.1, time instant t_c marks a scheduling event at which all three task jobs are released. According to the priority order, task T_1 has the highest priority and it is directly assigned to processor π_j for execution. At the same time instant $t_c = 0$, LBT is performed on other two lower priority tasks. Using equation 4.1, the anticipative laxity of task T_2 –i.e., l_2, in the presence of higher priority task T_1 is non-negative –i.e., l_2= $14-(0+6+8+0)$=0. This result shows that, at time instant $t_c = 0$, task T_2 can still meet its deadline if it will be executed *after* the worst-case completion of T_1. Thus, T_2 can be *deferred* at this stage on processor π_j –i.e., $T_2[\pi_j]$ at time $t_c = 0$. Similarly, the anticipative laxity of task T_3 –i.e., l_3, is calculated in the presence of higher priority tasks T_1 and T_2 such that, l_3= $20-(0+5+8+6)$>0. Like T_2, T_3 can also meet its deadline after the worst-case completion of T_1 and T_2 and therefore, can be deferred until next scheduling event occurs. Note that tasks T_2 and T_3 are shown dotted because this is only an *anticipation* of the future schedule of tasks at current scheduling event that may change upon the arrival of next event.

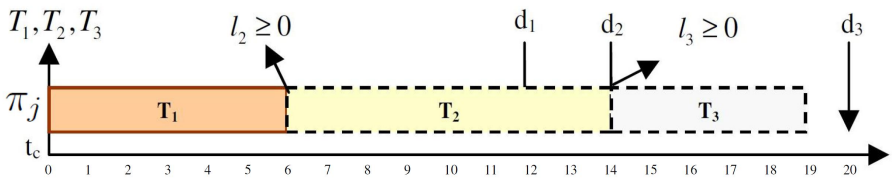


Figure 4.1: Laxity Bottom Test (LBT) using anticipative laxity l_i.

4.3.2 Working Principle

The working principle of AsDPM is illustrated in algorithm 5. As stated earlier in section 4.3.1, at every scheduling event, AsDPM performs laxity bottom test starting with the assumption that at most one processor is running to accommodate most of the workload and gradually increases the computational resources. Variable j in algorithm 5 refers to the number of processors. Upon the arrival of a scheduling event, all task queues (TQ, RuTQ, ReTQ, and DeTQ) are updated and sorted in accordance with the priority order of tasks specified by the governing scheduling algorithm (lines $1-3$). Once all task queues are sorted, highest priority j task(s) from ReTQ are assigned to j processor(s) (line 5). For rest of the ready tasks present in ReTQ, LBT is performed considering the first target processor (line $6-9$). If a task passes LBT, it is moved into DeTQ –i.e., it is deferred from execution at current scheduling event. Otherwise, if a task does not pass LBT then it implies that currently available running processors are not sufficient to satisfy the concurrent resource requirement of ready tasks and some tasks may miss their deadline in future. In this case, all tasks which are deferred or running –i.e., present in RuTQ or DeTQ, are put into ReTQ again and more processors are activated. This procedure is repeated until ReTQ becomes empty –i.e., until all tasks present in ReTQ are either moved to RuTQ or DeTQ.

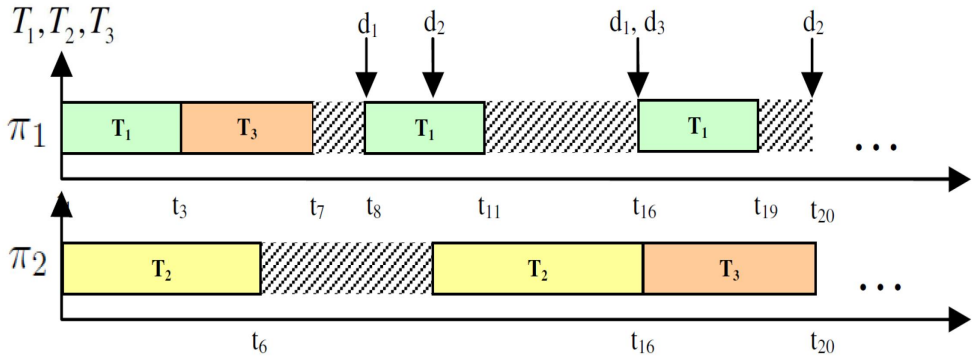
Note that when more than one processors are active, a task has to pass its LBT on *at least* one processor. For instance, with two processors considered as running at a particular scheduling event, if a task has its LBT failed on first processor, let us say π_j, but passed on the other processor, let us say π_{j+1}, then the task is deferred on π_{j+1} and next task from ReTQ is taken for test. Also note that, deferring of tasks is valid only between any two successive scheduling events. Upon the arrival of next scheduling event, the same process repeats itself and as a result, the number of active processors may change.

Let us consider another simple example that demonstrates the working principle of AsDPM.

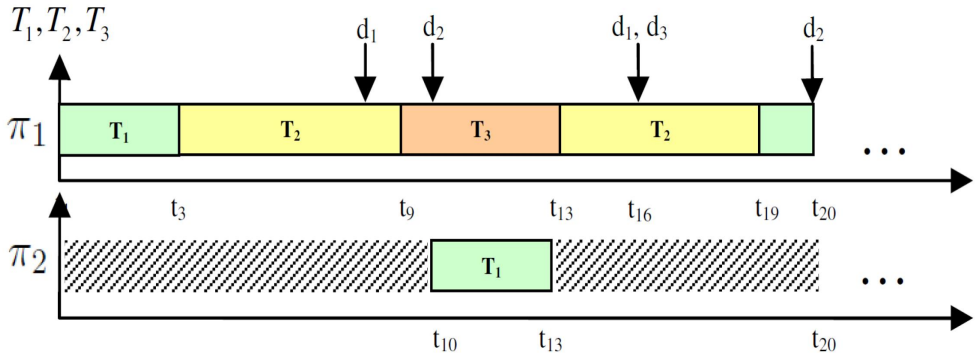
Example–2: Let us consider a task set composed of three periodic, independent, and synchronous tasks such that τ=$\{T_1, T_2, T_3\}$. Th quadruplet values for T_1, T_2, T_3 are $\{0, 3, 8, 8\}$, $\{0, 6, 10, 10\}$, and $\{0, 4, 16, 16\}$, respectively. Task set τ is scheduled over two processors under global EDF scheduling algorithm. Figure 4.2(a) illustrates the schedule of τ using global EDF scheduler. In this schedule, some random idle time intervals (gray dashed area) are generated due to the priority mechanism of EDF. These idle time intervals, when considered individually, may not be long enough as compared to BET for transitioning processors into power-efficient state. Figure 4.2(b) illustrates the same schedule using global EDF scheduler with AsDPM in place. It can be noticed in figure 4.2(b) that all idle time intervals from processor π_1 are extracted and clustered on π_2 to elongate the idle time on π_2. It becomes much easier to transition processor π_2 to power-efficient state after elongating idle time interval.

Algorithm 5 *Assertive* Dynamic Power Management

```
1: assign  j = 1
2: for each scheduling event do
3:   sort  TQ, ReTQ, RuTQ, and DeTQ w.r.t. scheduler's priority order
4:   repeat
5:     move highest priority j task(s) from ReTQ to RuTQ
6:     for every remaining task i in ReTQ do
7:       if  l_i ≥ 0 on j processor(s) then
8:         move  T_i to DeTQ
9:       else
10:        move  all tasks from DeTQ and RuTQ to ReTQ
11:        assign  j = j + 1
12:        activate  j processors
13:  until ReTQ is empty
```



(a) Execution trace of τ using global EDF scheduler



(b) Execution trace of τ using global EDF scheduler in the presence of AsDPM.

Figure 4.2: Schedule of τ using global EDF scheduler. (a) Without AsDPM. (b) With AsDPM.

In algorithm 5 (line 12), it is mentioned that additional processor(s) should be activated if the anticipative laxity of any low priority task results in negative value –i.e., that task will miss its deadline for its currently released job while using the available platform resources. In other words, the number of processors running at current scheduling event are insufficient to fulfill the execution requirement of all released tasks and therefore, more processor(s) should be brought to running state. However, a processor cannot be immediately recovered from power-efficient state due to temporal penalties associated with state-transition. The AsDPM technique solves this issue by using its a priori (online) knowledge of the increased platform resource requirement, thanks to anticipative laxity. From the definition of anticipative laxity, we know that it does not represent the real urgency or real laxity of a low priority task to execution. Rather, it makes a futuristic projection of resource requirement in order to ensure deadlines of all released jobs with currently available platform resources. Due to this futuristic projection, a processor has time to recover when it is required. Since the highest priority j task(s) do not pass LBT and are assigned immediately to j processor(s) (line 5), therefore, anticipative laxity of *only* lower priority tasks is tested. Hence, only lower priority tasks can potentially miss deadlines if their anticipative laxity results negative and if the number of running processor is not increased. By the time these lower priority tasks really start executing, additional processor(s) can be recovered from power-efficient state. Thus, the recovery time of a processor from power-efficient state to running state can be *masked* with the execution of higher priority executing task(s) before the time a processor is *actually* required to be functional. In the following, we describe how AsDPM deals with some corner cases.

Case 1: *A processor's recovery time from power-efficient state to running state might be greater than the absolute laxity (L_i) offered by a task T_i at its release instant.*

Let us consider that a lower priority task T_i is released at any point in time. At this time instant, τ^{deff} is empty –i.e., there is no higher priority deferred task and all the higher priority task (than T_i) are currently running. Let us suppose that the anticipative laxity of T_i results negative immediately after it is released. This implies that, T_i will miss its deadline on all processors currently running. Hence, it immediately requires additional computing resources to meet its deadline. In this scenario, if a processor's recovery time is larger than both the absolute laxity (L_i) of T_i and the remaining execution time (C^{rem}) of currently executing higher priority tasks then T_i will miss its deadline. To avoid such scenario, we propose two modifications.

1. Use a maximum value between remaining execution time (C_j^{rem}) of currently executing higher priority task on processor π_j and processor's recovery time (t_{trans}) from the most power-efficient state.

$$C_{max}^{rem} = max\left(C_j^{rem}, t_{trans}\right) \tag{4.2}$$

$$l_i = d_i - \left(t + C_{max}^{rem} + C_i^{rem} + \sum_{T_k[\pi_j] \in \tau^{deff}} C_k^{rem}\right) \tag{4.3}$$

2. Target application tasks should not have their absolute laxity (L_i) smaller than the recovery time (t_{trans}) of target processors from their most power-efficient state.

Case 2: *Release of an intermediate priority task might cause an already deferred higher priority task to miss its deadline.*

Let us consider an example task set composed of four periodic tasks ($n = 4$), such that τ=$\{T_1, T_2, T_3, T_4\}$. This task set is scheduled over two processors ($m = 2$) such that Π=$\{\pi_1, \pi_2\}$ under global EDF scheduler. At time instant t_c, tasks T_1, T_2, and T_4 are released. According to the working principle of AsDPM, task T_1 starts executing immediately on π_1 and T_2 and T_4 were deferred on π_1 as illustrated in figure 4.3(a). At this stage, one processor seemed sufficient to meet deadlines of all released tasks. Let us consider that an intermediate priority task T_3 is released at time instant t_{c+1} and invokes the scheduler again as illustrated in figure 4.3(b). At this scheduling event, scheduler updates the priority order of all released tasks. AsDPM performs LBT on all sorted tasks present in ReTQ as mentioned in the algorithm 5. Due to updated priority order, task T_4 fails the LBT which implies the requirement of additional processors to meet deadlines of all tasks. Thus, processor π_2 should be activated at time t_{c+1}. In figure 4.3(b), red rectangular box represents the recovery time t_{trans} of processor π_2.

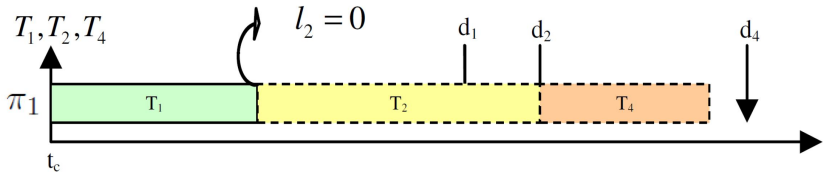
Since AsDPM preserves the order of execution of tasks, therefore, the newly activated processor π_2 must execute T_2 ahead of T_3 or T_4 as it is the next highest priority task in ReTQ. However, at this point in time, the runtime laxity of T_2 is decreased from its absolute laxity L_i due to earlier deferring and it cannot meet its deadline if it is forced to wait for execution on π_2 processor. To avoid this situation, the real laxity of highest priority deferred task, which is T_2 in this case, is compared with the newly activated processor's recovery time, which is π_2 in this case, and if the resultant value is positive then T_2 is assigned to π_2. Otherwise, immediate next priority task after T_2 is assigned to π_2 processor –i.e., T_3 is assigned to π_2. Note that the order of execution of tasks does not change as T_2 start executing on π_1 before T_3 starts executing on π_2.

Case 3: *Energy inefficient state transitions might occur on processors due to varying platform resource utilization.*

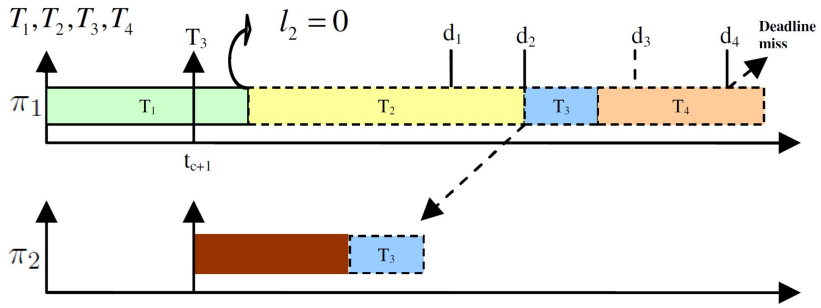
Every scheduling event has an impact on the concurrent resource utilization of the system. For instance, release of a new job potentially increases concurrent platform resource utilization while completion of a job potentially decreases it. However, the exact impact of future scheduling events on the overall platform resource utilization of the system is not known a priori to the scheduler. This uncertainty may cause AsDPM to be *over-optimistic* in the sense that it would extract all idle intervals in the beginning and might require more processing power to meet deadlines later. This over-optimism can be avoided for periodic task models for which, knowing the future/upcoming scheduling event is possible thanks to recurring nature of tasks. While using AsDPM, the information concerning the type of upcoming scheduling event is extracted from sorted TQ, ReTQ and RuTQ. For implicit deadline tasks, the deadline of a task also refers to the release event of its next job and therefore, it is sufficient to compare the deadlines of highest priority tasks present in TQ, ReTQ, and RuTQ to determine the earliest future release event using equation 4.4.

$$r^{next} = min\left(d_{TQ}, d_{ReTQ}, d_{RuTQ}\right) \tag{4.4}$$

Here, d_{TQ}, d_{ReTQ}, and d_{RuTQ} refer to the deadlines of highest priority tasks present in TQ, ReTQ, and RuTQ, respectively. The knowledge of next release instant of any upcoming job(s) is helpful for AsDPM in determining when exactly the platform resource utilization might increase. Consequently, the idle intervals which occur very close to r^{next} are not



(a) Projected schedule of tasks at time t_c without intermediate priority task T_3.



(b) Projected schedule of tasks at time t_{c+1} with intermediate priority task T_3.

Figure 4.3: Impact of an intermediate priority task's release. (a) Projected schedule of tasks at time t_c without intermediate priority task T_3. (b) Projected schedule of tasks at time t_{c+1} with intermediate priority task T_3.

extracted. Thus, state transitions on processors which might result is an energy loss are avoided. The closeness to r^{next} is user-specified. For instance, it can be equal to BET of the processor.

4.3.3 Choice of Power-efficient State

Modern processors support multiple power-efficient states. Since, there are temporal and energy penalties associated with state transitions, therefore, a processor needs to be put in the power-efficient state long enough to save energy –i.e., at least equal to the break-even time (BET). Under AsDPM, some processors of the platform have larger workload while others have less workload. For those processors having larger workload, and consequently shorter idle time intervals, it is not so beneficial, some times even penalizing, to transition them into deeper power-efficient states. This is because the number of transitions on such processors is greater and accumulates large transition cost. In addition, generally, the more a state is power-efficient, the more it takes (time and energy) to recover a processor from that state. However, for other processors having longer idle time intervals, it is advantageous to put them in more power-efficient states as they are not often recovered to running state. Once the AsDPM technique has extracted idle time intervals, processors are then assigned suitable power-efficient state with respect to their worst-case workload. Processors, from which idle time intervals are mostly extracted at runtime, are either assigned no power-efficient state at all or they are assigned state having least recovery time penalty. Other processors, on which idle time intervals are accumulated at runtime, are assigned more power-efficient states.

4.4 Static Optimizations using AsDPM

In section 4.3, it has been mentioned that AsDPM itself is an online admission control technique which decides when exactly a ready task shall execute. Without this admission control, all ready tasks are executed as soon as there are enough computing resources (processors) available in the system. This feature of AsDPM not only helps optimizing the use of platform resources in an online fashion, but also can be used for static/offline optimizations of platform resources. By optimized platform resources, we mean the minimum number of required processors under worst-case workload and the corresponding operating voltage-frequency level of processors which can result in an optimal architecture configuration from energy consumption point of view. All timing constraints of real-time tasks must always be respected by the resultant architecture configuration.

The static optimal architecture configuration is determined for worst-case workload requirements of a target application. Note that this configuration may vary from one target application to another. The process of determining static optimal architecture configuration is based on simulations under worst-case application workload. The number of processors in the target processing platform are not specified a priori (in simulations only). The scheduler, by using AsDPM, can activate as many processors as required to meet timing constraints of target application at runtime. For every voltage-frequency level supported by the target processors, starting from the maximum voltage-frequency level, simulation of one complete frame (i.e., the first hyper-period) of target application tasks is performed under worst-case workload. As a result, the number of processors used in the simulated trace and the energy consumed are obtained. The number of processors used and corresponding voltage-frequency level would be considered as a static architecture configuration which is not necessarily the static *optimal* configuration. The process is repeated for every voltage-frequency level. This is similar to static voltage and frequency scaling (SVFS) process with the exception that the number of processors are not fixed a priori. Once all configurations for different voltage-frequency levels are obtained, the configuration consuming least energy while respecting physical constraints of platform architecture (i.e., maximum allowable number of processors in real platform) and timing constraints of target application is chosen to be the static optimal architecture configuration. Note that no power-efficient state is assigned at this stage to any processor if it remains idle. In section 4.5, we shall elaborate further how static optimal architecture configuration is achieved through simulations.

4.5 Experiments

In this section, we provide the reader the evaluation of the AsDPM technique, which is based on simulation results. The performance of AsDPM is evaluated using STORM simulator (Simulation TOol for Real-time Multiprocessor Scheduling) [108]. We consider the same system model –i.e., task model, processing platform, and power and energy models, as presented in chapter 2. Simulations are carried out using global EDF scheduling algorithm.

4.5.1 Target Application

H.264 video decoder application is taken as main use case application. H.264 is a high compression rate algorithm [88] relying on several efficient techniques extracting spatial (within a frame) and temporal dependencies (between frames). The main steps of the H.264 decoding process are depicted in figure 4.4 and consist in the following:

1. A compressed bit stream coming from the NAL layer is received at the input of the decoder.
2. Data are entropy decoded and sorted to produce a set of quantized coefficients.
3. These coefficients are then inverse quantized and inverse transformed.
4. Data obtained are then added to the predicted data from the previous frames depending upon the header information.
5. Finally the original block is obtained after the de-blocking filter to compensate the block artifacts effect.

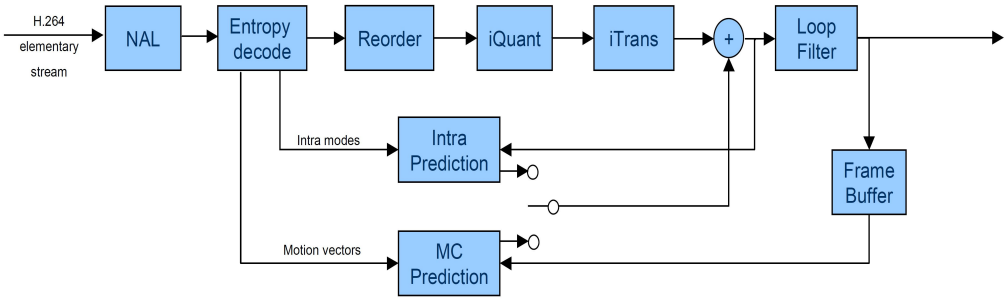


Figure 4.4: Block diagram of H.264 video decoding scheme.

The H.264 video decoder application can be broken down into various tasks sets corresponding to different types of parallelization. In our experiments, we use two different task models of H.264 proposed by Thales Group, France [115] in the context of French national project Pherma [86]. The proposed task models are:

- H.264 slices version
- H.264 pipeline version

In both versions, the defined timing constraint is that an output frame must be produced, lets say every X ms, where X is a Quality of Service (QoS) level set by the user. In the following, each version is discussed briefly.

4.5.1.1 H.264 slices version

The main characteristic of this version is that the algorithm is parallelized on the slices of the frame as illustrated in figure 4.5. This parallelization is of SPMD (Single Program Multiple Data) shape since the same algorithm is performed on different data. For this version, it is considered that frames are made up of 4 slices. Since slices inside a frame can be computed independently, therefore, one task is assigned for each slice to be computed. Thus, four tasks, named *slice-processing*, can run simultaneously in this version. There are some synchronizations required between tasks that must be handled to ensure a proper processing without data corruption. These synchronization are handled through the task named *SYNC* on the left hand side in figure 4.5. At the beginning of each new frame, tasks can access only sequentially to the input data buffer. Therefore, there is a slight

overhead in the real beginning of each start up of the task named *Slice*. This behavior is due to the access of shared resource which is protected by a semaphore. Due to temporal dependencies between frames, it is not possible to compute the next frame if the previous one has not been completely decoded. Thus, at the end of each slice computation, tasks need to be resynchronized using task named *SYNC* (on the right hand side in figure 4.5). As a result, input data must be present and the previous frame must be decoded at the start of decoding a new frame. Hence, H.264 slices version comprises of seven periodic tasks as shown in table 4.1. All values for B_i and C_i are given at maximum frequency of PXA270 processor (i.e., 624-MHz).

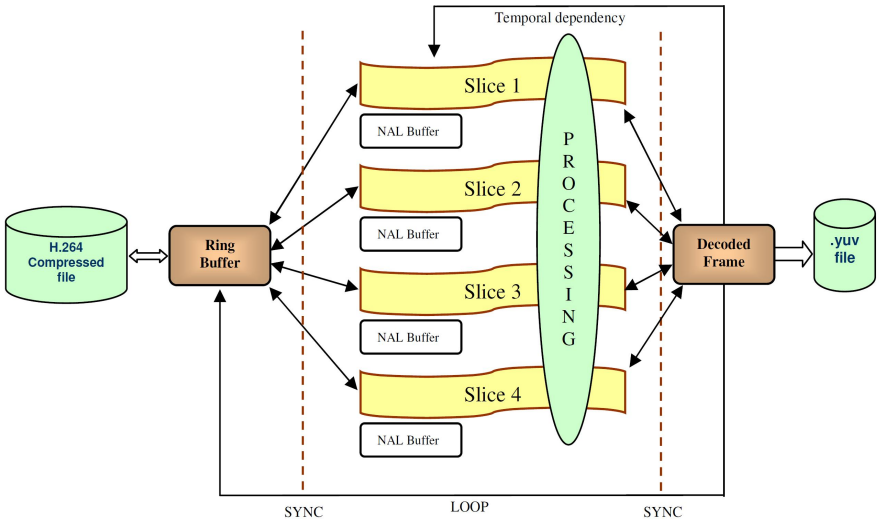


Figure 4.5: Block diagram of H.264 decoding scheme slices version.

Table 4.1: H.264 video decoder application task set for slices version

Task Name	r_i	C_i	B_i	d_i	P_i
NEW-FRAME (T_1)	0	1	1	40	40
NAL-DISPATCH (T_2)	0	2	1	10	10
SLICE1-PROCESSING (T_3)	10	42	21	120	120
SLICE2-PROCESSING (T_4)	20	42	21	120	120
SLICE3-PROCESSING (T_5)	30	42	21	120	120
SLICE4-PROCESSING (T_6)	40	42	21	120	120
REBUILD-FRAME (T_7)	160	2	1	120	120

4.5.1.2 H.264 pipeline version

Main characteristics of the pipeline version are that the whole processing has been pipelined into seven tasks, namely; TG, NA, SI, RE, LI, and RA. Figure 4.6 illustrates the order of execution for these tasks. The task related to the image rendering process *RE* has also been parallelized into sub tasks. In the physical implementation of pipelined version provided by

Thales Group, France [115], synchronization between tasks are handled through a blocking FIFO communication mechanism -i.e., tasks are blocked when the FIFO is empty or full or also when there is not enough data to be read inside the FIFO. In this version, the only constrain to start the decoding of a new frame is the availability of the input data. Once new data are available, another pipeline stage can start. In simulation, we extract this sequential execution of tasks by adding an offset at each periodic task's release. With the help of this offset, tasks process different data at different pipeline depth and input data are assumed to be always available. Table 4.2 gives the resulting task set for H.264 video decoder pipeline version with offsets at release of tasks.

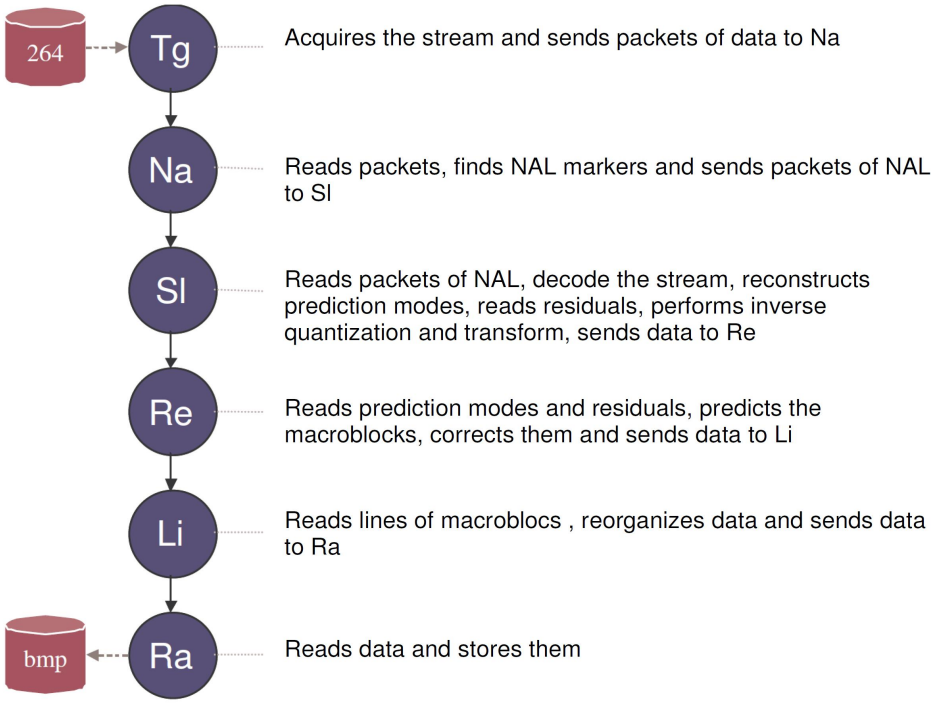


Figure 4.6: Block diagram of H.264 decoding scheme pipeline version.

Table 4.2: H.264 video decoder application task set for pipeline version

Task Name	r_i	C_i	B_i	d_i	P_i
TG (T_1)	0	2	1	15	15
SI (T_2)	15	3	2	15	15
RE-1 (T_3)	30	17	8	30	30
RE-2 (T_4)	30	17	8	30	30
RE-F (T_5)	60	8	4	30	30
LI (T_6)	90	3	2	30	30
RA (T_7)	120	2	1	30	30

4.5.2 Simulation Results

Simulation results for both versions of H.264 video decoder application are provided using different throughput requirements. Throughput requirement is a user-specified parameter, measured as frames per second (fps), which determines different levels of Quality of Service (QoS). We demonstrate that how static optimal architecture configuration for different frame rates is obtained using AsDPM.

4.5.2.1 Determining static optimal architecture configurations

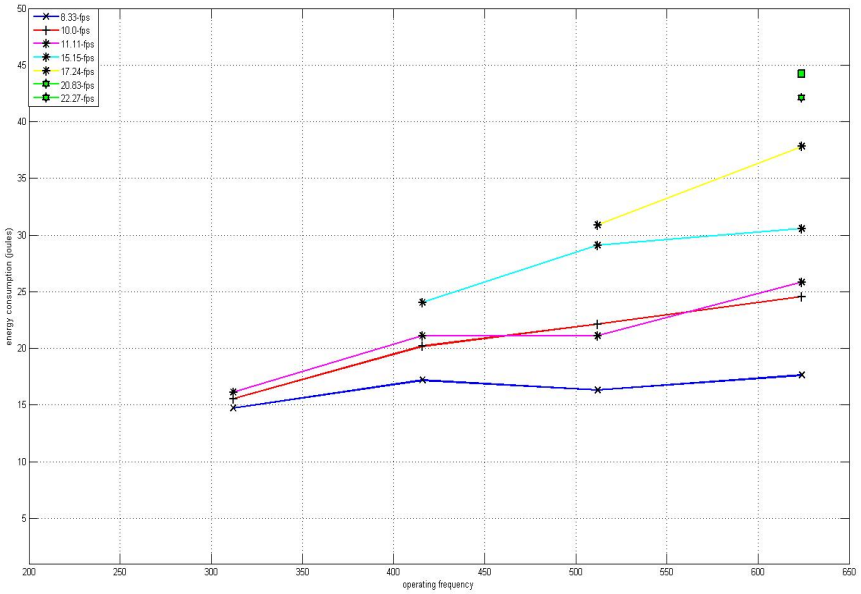


Figure 4.7: Simulation results on the changes in energy consumption for H.264 video decoder application (slices version) for various frequencies.

As mentioned in section 4.4, static optimal architecture configuration is obtained through simulations on every voltage-frequency level supported by target processors using worst-case execution requirement of all tasks. For both H.264 slices and pipeline version, we carry-out simulations at different quality of service requirements –i.e., frames per second. Table 4.3 refers to the results we obtain for H.264 slices version. These results are obtained for a simulation trace of 10-seconds. In the following, we analyze these results.

We observe that in order to meet the timing constraints of all tasks, e.g., for throughput rate of 8.33-fps with an operating voltage-frequency level of (1.55V, 624MHz), there must be at least 3 processors. Energy consumption in this case will rise up to 17.6-joules. Note that, based on these results, the static optimal architecture configuration seems to be with 6 processors at (1.25V, 312MHz) with 14.719-joules for a throughput rate of 8.33-fps as shown in table 4.3. However, if the operating voltage-frequency level is decreased below (1.25V, 312MHz), it will not be possible to meet the timing constraints of tasks as mentioned in the

Table 4.3: Static architecture configurations for H.264 video decoder slices version

Frame Rate(FPS)	Frequency (MHz)	Processors (m)	Energy (Joules)	Deadline Miss
8.33	624	3	17.630	NO
	520	3	16.319	NO
	416	5	17.188	NO
	312	6	14.719	NO
	208	7	8.979	YES
	104	–	–	–
10.0	624	4	24.553	NO
	520	5	22.116	NO
	416	6	20.185	NO
	312	6	15.554	NO
	208	7	8.980	YES
	104	–	–	–
11.11	624	4	25.851	NO
	520	4	21.098	NO
	416	6	21.117	NO
	312	6	16.110	NO
	208	7	8.975	YES
	104	–	–	–
15.15	624	4	30.551	NO
	520	6	29.101	NO
	416	6	24.036	NO
	312	7	15.724	YES
	208	–	–	–
	104	–	–	–
17.24	624	6	37.826	NO
	520	6	30.885	NO
	416	6	19.466	YES
	312	–	–	–
	208	–	–	–
	104	–	–	–
20.83	624	6	42.131	NO
	520	6	25.049	YES
	416	–	–	–
	312	–	–	–
	208	–	–	–
	104	–	–	–
22.27	624	6	44.247	NO
	520	6	25.901	YES
	416	–	–	–
	312	–	–	–
	208	–	–	–
	104	–	–	–

last column of table 4.3. An interesting observation on these results is that reducing voltage-frequency level might not necessarily always decrease overall energy consumption. For

Table 4.4: Static architecture configurations for H.264 video decoder pipeline version

Frame Rate(FPS)	Frequency (MHz)	Processors (m)	Energy (Joules)	Deadline Miss
10.0	624	1	9.204	NO
	520	1	6.716	NO
	416	2	6.458	NO
	312	2	5.349	NO
	208	6	7.362	NO
	104	5	2.703	YES
12.0	624	1	9.211	NO
	520	2	8.785	NO
	416	2	6.721	NO
	312	2	5.520	NO
	208	3	3.809	NO
	104	6	3.281	YES
15.0	624	1	9.224	NO
	520	2	9.600	NO
	416	2	8.514	NO
	312	3	7.611	NO
	208	5	6.317	NO
	104	6	3.301	YES
20.0	624	2	11.982	NO
	520	2	10.402	NO
	416	2	8.561	NO
	312	3	7.848	NO
	208	5	6.367	YES
	104	–	–	–
25.0	624	2	14.631	NO
	520	2	12.665	NO
	416	3	11.327	NO
	312	3	8.048	NO
	208	4	5.111	YES
	104	–	–	–
32.0	624	2	17.768	NO
	520	3	16.860	NO
	416	4	15.651	NO
	312	5	11.589	YES
	208	–	–	–
	104	–	–	–

instance, static architecture configuration at (1.45V, 520MHz) with 3 processors consumes less energy than (1.35V, 416MHz) with 5 processors. As a result, the increase in number of processor must be followed by an sufficient decrease in the operating voltage-frequency level, otherwise, the overall energy consumption will not be lowered enough. For each throughput requirement, the optimal configuration from the energy saving point of view is highlighted in red color in table 4.3. Note that an optimal or most suitable static architecture configuration depends also on the available physical platform. Figure 4.7 graphically illustrates the results presented in table 4.3 for those frequencies only, for which no deadline is missed. Figure

4.7 illustrates that reducing operating frequency does not always result in a linear decrease in energy consumption.

Table 4.4 represents similar results on static architecture configurations for H.264 pipeline version for different throughput rates. Note that with pipeline version (see table 4.2), the achievable throughput rate is up to 32-fps. Configurations highlighted in red color against each throughput requirement are the optimal configurations only from energy consumption point of view. Figure 4.8 illustrates the difference in energy consumption between a statically non-optimized EDF schedule and a statically optimized EDF schedule of H.264 video decoder application (slices version) using AsDPM. Similar results are presented in figure 4.9 for pipeline version of H.264 video decoder application. Energy consumed under statically non-optimized EDF-based schedule refers to the energy consumption at maximum voltage-frequency level –i.e., (1.55V, 624MHz), and m processors as required by the schedulability analysis of EDF for every QoS requirement.

The use of AsDPM for determining static optimal architecture configurations enables the designer to explore target architecture. Indeed, we have been able to test the timing constraints of target application at different throughput requirements. Moreover, we have been able to identify what would be the minimum needs in terms of processing platform resources (i.e., number of processors and their corresponding operating voltage-frequency level) to fit different throughput requirements. If the end-user has an energy budget to respect, this technique helps him/her to have an energy forecast of the target configuration.

Table 4.5: Static *optimal* architecture configurations for H.264 video decoder slices version for different QoS requirements

Frame Rate (FPS)	Frequency (MHz)	Processors (m)	Energy (Joules)
8.33	312	6	14.719
10.0	312	6	15.554
11.11	312	6	16.110
15.15	416	6	24.036
17.24	520	6	30.885
20.83	624	6	42.131
22.27	624	6	44.247

Table 4.6: Static *optimal* architecture configurations for H.264 video decoder pipeline version for different QoS requirements

Frame Rate (FPS)	Frequency (MHz)	Processors (m)	Energy (Joules)
10.0	312	2	5.349
12.0	208	3	3.809
15.0	208	5	6.317
20.0	312	3	7.848
25.0	312	3	8.048
32.0	416	4	15.651

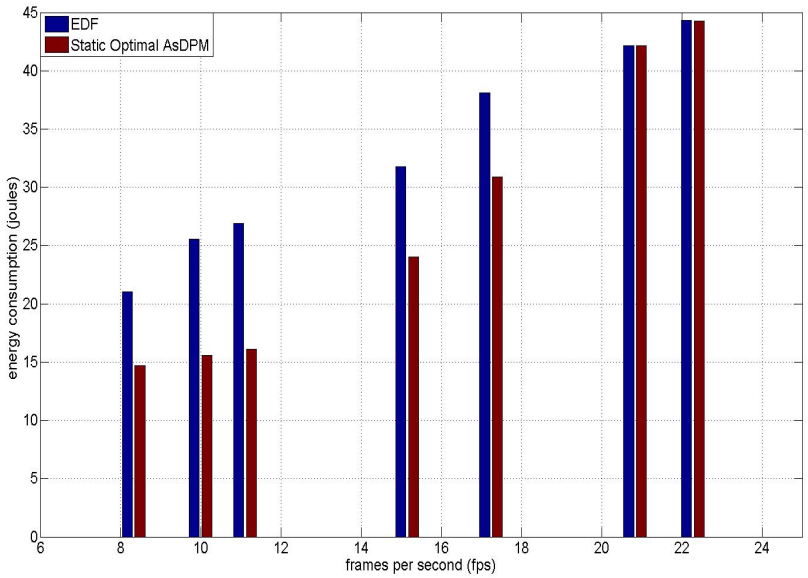


Figure 4.8: Simulation results on energy consumption under statically non-optimized EDF schedule and statically optimized EDF schedule using AsDPM for H.264 video decoder application (slices version).

4.5.2.2 Results using AsDPM as an online technique

Once the static optimal architecture configurations are obtained for different QoS requirements as presented in section 4.5.2.1, AsDPM can be applied as an online DPM technique on these configurations to further optimize energy consumption. Since real-time applications potentially exhibit variations in their actual execution time and generate dynamic slack, therefore, there are more opportunities to save energy at runtime.

To demonstrate the effectiveness of AsDPM as an online DPM technique, we simulate the task sets for slices and pipeline versions of H.264 video decoder present in table 4.1 and table 4.2, respectively, using static optimal architecture configurations present in table 4.5 and table 4.6, respectively. Note that the best-case to worst-case execution time ratio (bcet/wcet ratio) for each task is varied randomly between 50% and 100% of its wcet such that the actual execution time of each task has a uniform probability distribution function as suggested in [10]. Simulation results obtained using STORM simulator are presented in figure 4.10 for slices version and in figure 4.11 for pipelined version. Figure 4.10 illustrates the difference in energy consumption of non-optimized EDF schedule, statically optimized EDF schedule using AsDPM, and EDF schedule using online AsDPM. It can be noticed that the online AsDPM technique further increases energy gains of statically optimized schedule for more or less every QoS requirement. Energy gains obtained by the online AsDPM technique reach from 14% up to 30% compared to the results presented in table 4.5. In the best-case (for QoS requirement of 8.33-fps), energy savings are observed up to 32.7%. Similar results are presented in figure 4.11. Energy gains obtained by the online AsDPM technique for H.264 pipelined version range from 11% to 35% as compared to statically optimized results presented in table 4.6 for different QoS requirements. The best-case

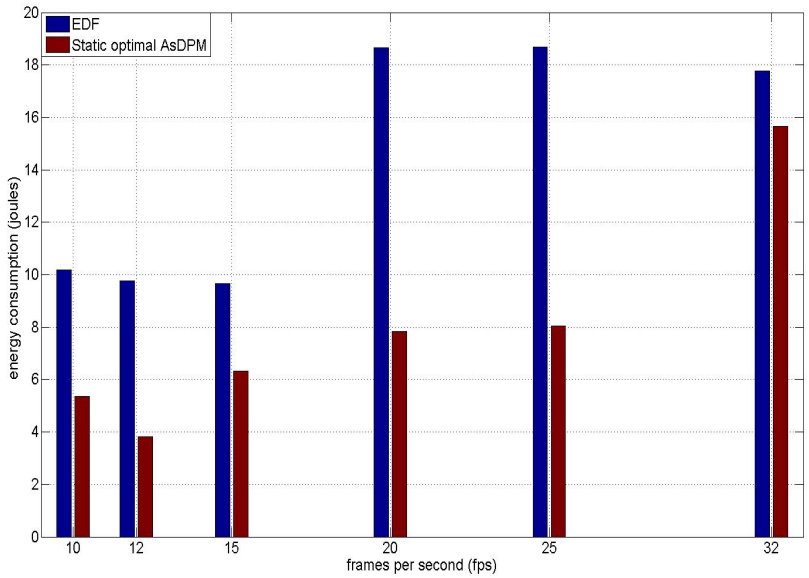


Figure 4.9: Simulation results on energy consumption under statically non-optimized EDF schedule and statically optimized EDF schedule using AsDPM for H.264 video decoder application (pipeline version).

energy savings in this case are observed up to 35.8% for QoS requirement of 15-fps.

Note that the results provided on the percentage of energy gains while using the online AsDPM technique are given as compared to the energy gains of statically optimized results. If no static optimizations are performed, achievable gains on energy using the online AsDPM technique alone would be significantly large as compared to the presented results.

4.5.3 Comparative Analysis of the AsDPM Technique

Although, many of the existing DPM techniques claim significant power and energy savings for real-time application models, producing the same results for such techniques in simulation settings other than the one used by authors is neither straightforward nor trivial. This is one of the limitations for us as well while comparing the performance of our proposed technique with existing techniques. Nevertheless, in order to give a reasonable idea on the performance of our proposed technique, in this section, we provide a comparative analysis of energy savings under AsDPM and a theoretically *ideal DPM* technique. A theoretically ideal DPM technique is the one that would have no temporal or energy penalty while transitioning a processor to power-efficient states and it could transition a processor as soon as an idle time interval is detected. Moreover, all idle time intervals could be exploited for energy savings under such technique. Although, it is impossible to have zero penalties in existing processor technology, energy-efficiency of such an ideal DPM technique gives a good theoretical reference for measuring the performance of other techniques.

Figure 4.12 illustrates the results we obtain while comparing the energy savings under non-optimized EDF schedule, static optimal AsDPM, online AsDPM, and ideal DPM for

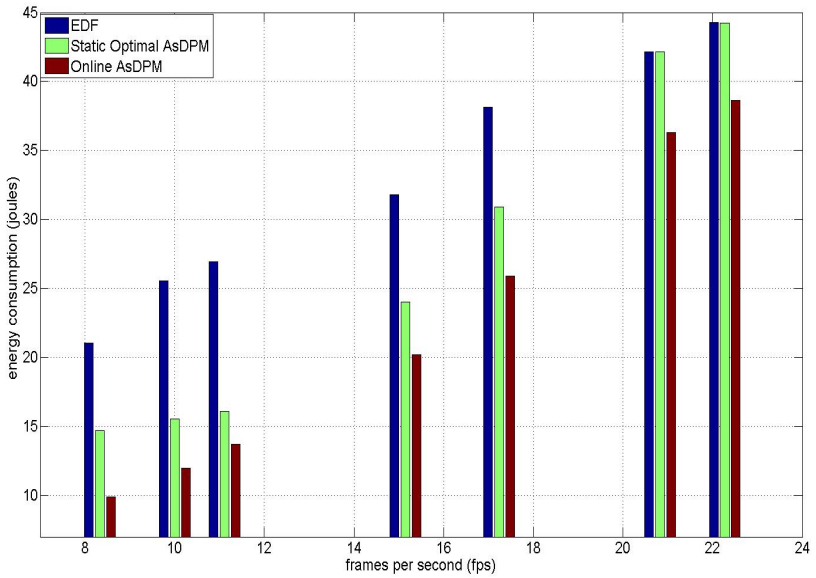


Figure 4.10: Simulation results on the energy consumption under statically non-optimized EDF schedule, statically optimized EDF schedule using AsDPM, and EDF schedule using online AsDPM for H.264 video decoder application (slices version).

the slices version of H.264 video decoder. We have measured the energy gains obtained by statically optimized EDF schedule of tasks using AsDPM within 50% to 55% of ideal DPM technique, and within 8% to 16% of ideal DPM technique while using the online AsDPM technique with EDF scheduler for the same application. These results are encouraging and depict that AsDPM, when used for both static and dynamic energy optimization, can achieve significantly large percentage of energy-efficiency that is close to the best-possible energy savings. Obviously, it is impossible to achieve energy gains of an ideal DPM technique due to the realistic temporal and energy penalties involved in transitioning processor states. Thus, AsDPM performs well while taking into account the transition penalties.

4.6 Future Perspectives of the AsDPM Technique

In this section, we highlight some future perspectives of AsDPM that are mainly related to the system-wide energy-efficiency and thermal issues in multiprocessor systems. Authors in [58, 130, 27] have suggested that devices such as memory banks, flash drives and wireless network interfaces are pervasive in modern embedded systems which consume considerable amount of energy. Hence, limiting the scope of energy-efficient techniques only to processors may not result in considerable overall energy savings. Moreover, while battery operated embedded systems need to meet an ever-increasing demand for performance with longer battery life, high performance systems, on the other hand, contend with the issues of heating. Authors in [27, 78] highlighted that heating issue is becoming a crucial design criterion

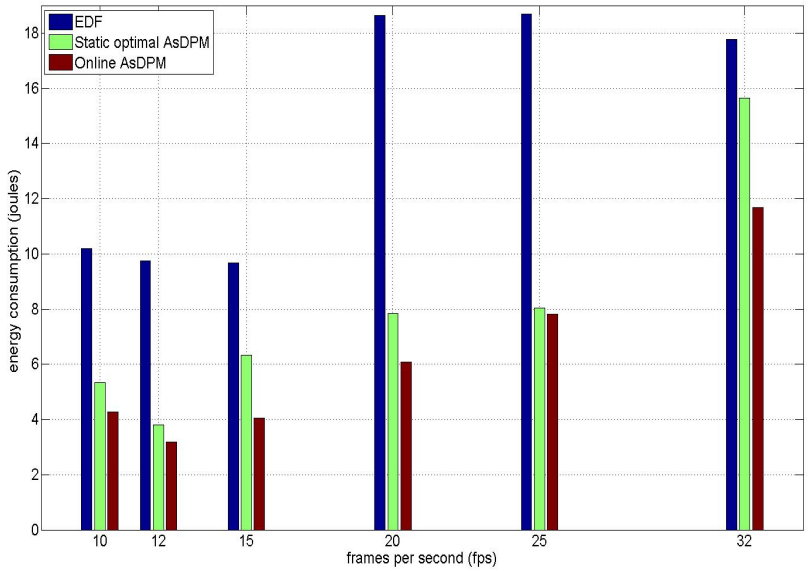


Figure 4.11: Simulation results on the energy consumption under statically non-optimized EDF schedule, statically optimized EDF schedule using AsDPM, and EDF schedule using online AsDPM for H.264 video decoder application (pipeline version).

in order to ensure QoS requirements and limit the performance degradation for multi-core and many-core systems with 2-dimensional and 3-dimensional chip manufacturing. In this section, we briefly describe how AsDPM can be useful for other system components, especially memory subsystem. Moreover, we describe how AsDPM confronts with heating issue.

4.6.1 Memory Subsystem

Authors in [81] report that multi-banking in static and dynamic RAM-based main memory architectures has been used as a popular method for reducing energy consumption. In multi-banking, memory space is divided into multiple banks, each of which can be controlled independently of the others. One example of this type of memory architecture is RDRAM [93] in which each bank can work with four power-efficient states –i.e., active, standby, nap, and power-down. A memory bank can service a read/write request only in the active mode. If it is not servicing a memory request, it can be placed into one of aforementioned power-efficient state. Results reported by many researchers [31, 32, 41, 67, 82, 81] suggest that significant reduction in energy consumption of memories is achieved through multi-banking approach. Authors in [81, 82] propose techniques based on clustering dynamically created data with temporal affinity in the physical address space such that the data occupies a small number of memory banks and remaining (unused) banks can be shutdown to save leakage power. There is a trade off between energy saving and performance penalty (i.e., resynchronization cost in terms of time and energy required to bring back a memory bank

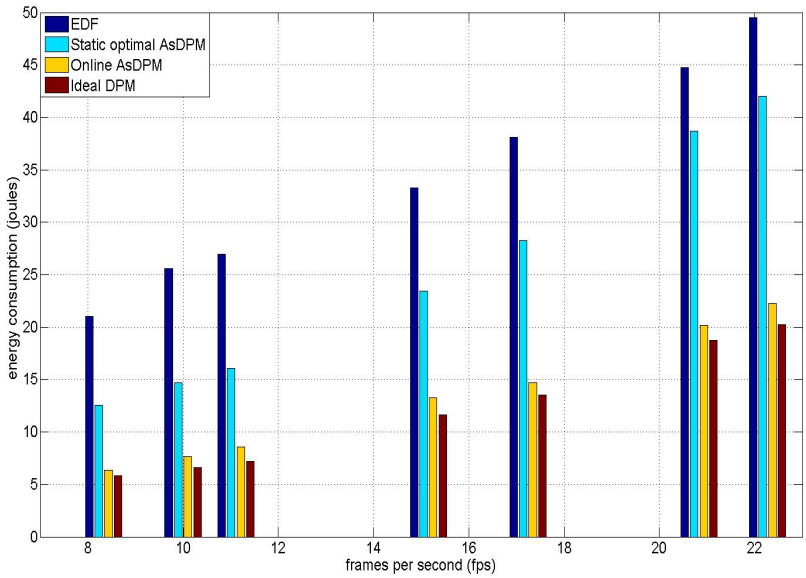


Figure 4.12: Simulation results on energy consumption of AsDPM in comparison with ideal DPM technique under the control of EDF scheduling algorithm for H.264 video decoder application (slices version).

to the active state) when selecting a power-efficient state to use for a given idle bank. The time between successive accesses to a given bank is the major determining factor in selecting the most suitable power-efficient state. Typically, larger the inter-access time of a bank, deeper the power-efficient state that can be applied and higher the energy saving of a state, higher the resynchronization cost.

We foresee that applying AsDPM (on processors) in systems which are based on multi-bank memory architectures can be potentially interesting. We consider a simplified multi-bank memory system in which each independent application task uses a separate memory bank and its data are not compacted using physical address space optimization techniques. However, all data and code concerning a single task can completely reside in one memory bank. For such system, the number of active memory banks would be equal to the number of tasks running at any time instant over a multiprocessor platform. Here, we recall from section 4.3.2 that AsDPM works as an admission control technique and seeks to optimize the number of running processors as much as possible (provided that no task misses its deadline) even if there is exploitable parallelism present in the application tasks. Eventually, resulting in an optimized minimum number of tasks (often less than total number of processors available in the platform) running at any given time instant. The working mechanism of AsDPM leads to reduce the number of active memory banks at runtime. That is, even if AsDPM is applied only on processors, energy consumption can be reduced at memory subsystem level by manipulating state of memory banks in accordance with the order of execution of tasks. Further energy savings can be obtained using physical address space optimization techniques –i.e, by putting multiple tasks in single bank (if possible) or allocating memory banks to various tasks in specific order so that some banks can remain

idle for longer duration.

Only for illustration purpose, we consider a simulation trace obtained from STORM simulator in which certain number of tasks are scheduled over three processors using global EDF scheduling algorithm. Figure 4.13(a) illustrates the case when AsDPM is not applied. In this figure, it can be noticed that the number of banks being used at different time instants vary quite frequently. Moreover, the number of state transitions is also significantly large. Whereas, in figure 4.13(b) which refers to the case when AsDPM is applied at processors, fewer than maximum number of memory banks are active most of the time and the number of transitions is reduced as well. We did not carry out detailed experiments to validate this observation due to some limitations with the simulation tool. However, we feel confident that with larger application task sets, the gains on energy at memory subsystem level can be capitalized.

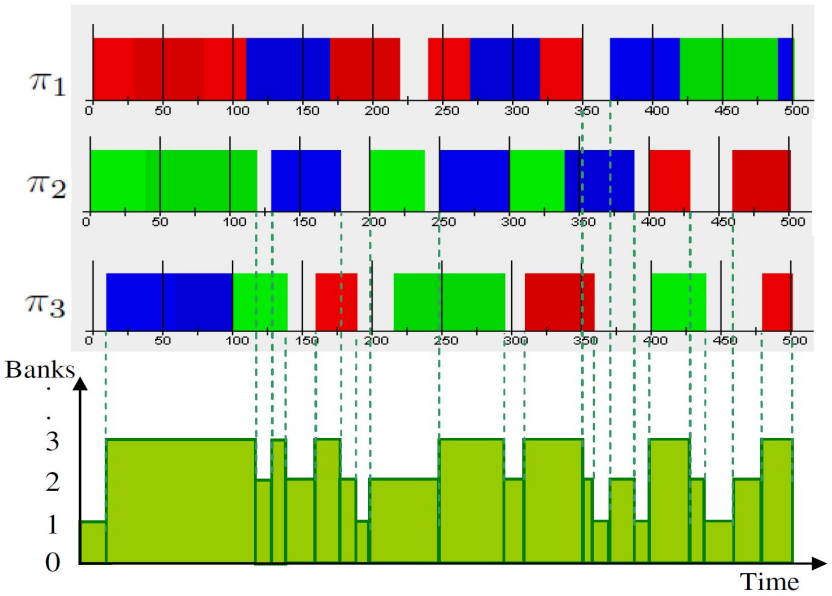
4.6.2 Thermal Load Balancing

Authors in [27, 78] have highlighted that, due to the scaling-down of transistor, the available chip surface for heat dissipation is reducing which results in increased power densities. Multiprocessor systems also behave as multiple heat sources which increase the likelihood of temperature variations over shorter time and chip area rather than just a uniform temperature distribution over entire die.

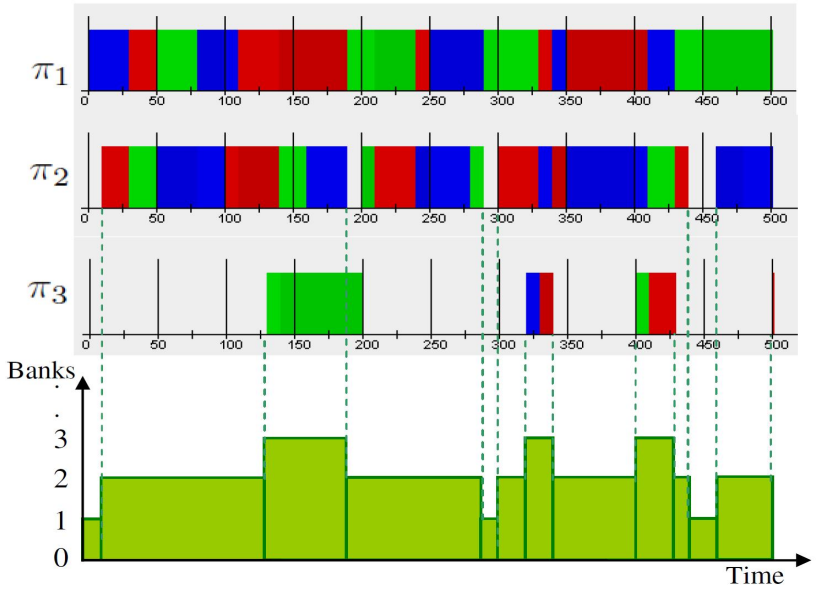
In fact, in its actual state, AsDPM can potentially cause thermal imbalance on processors of the system. The source of (potential) thermal imbalance in the system is the working principle of AsDPM, which seeks to minimize the number of running processors. That is, processors that are in running state are intended to be continuously running while processors in power-efficient state are supposed to remain in their state as long as possible, resulting in increased temperature for the running processors and eventually reduce life time. To address this issue, we consider the thermal model of microprocessors presented by authors in [73]. Authors state that, once the energy consumption of a processor is known, it can be used as an input for thermal model of processors. Using such a model, the temperature of processor can be estimated quite accurately. The temperature follows power, but does not change at once, rather it increases exponentially over time. The course of processor's temperature over time is modeled by authors in [73] through equation 4.5.

$$\upsilon(t) = \frac{-\widetilde{C}}{C_2} \cdot e^{-C_2 t} + \frac{C_1}{C_2} \cdot Pwr + \upsilon_0 \tag{4.5}$$

Here, υ_0 is the ambient temperature. C_1 and C_2 are constants depending on the thermal resistance and the thermal capacitance of processor and heat sink. $\widetilde{C}$ is an integration constant depending on the initial temperature of processor. Equation 4.5 consists of a dynamic part (first addend) and a static part (second and third addend). Static part depends only on the ambient temperature and power consumption of processor. It models the amount of temperature that a processor reaches after operating with constant power for long time. The dynamic part is an exponential function with time constant C_2 and models the exponential rise or decay of a processor's temperature after a change in power consumption. C_2 can be determined by starting a computation-intensive task, which produces a maximum of heat on a processor formerly idle, recording the temperature values over time, and fitting an exponential function to the experimental data. With knowledge of C_2, the constant C_1 can then be determined from the static part of equation 4.5 by measuring temperature against known power consumption of the processor. Note that this thermal model does not take into account the physical placement of multiple cores which can potentially create thermal hot-spots.



(a) Energy consumption of multi-bank memory under global EDF schedule without AsDPM.



(b) Energy consumption optimization of multi-bank memory under global EDF schedule using AsDPM.

Figure 4.13: Energy consumption in memory subsystem using multi-bank architecture. (a) Energy consumption of multi-bank memory under global EDF schedule without AsDPM. (b) Energy consumption optimization of multi-bank memory under global EDF schedule using AsDPM.

Equation 4.5 links the thermal variations of processor with the power consumption variations. Since power consumption can be estimated very precisely using many simulation tools, therefore, thermal variations of processor can be approximated using the model presented by [73]. The thermal imbalance in AsDPM can be dealt with using this model by maintaining a power consumption profile, which could be approximated to temperature variations of processors, and evaluate this profile at suitable time instants, such as every scheduling event. Based on the evolution of temperature over time, AsDPM can suggest the controlling scheduler to replace the heated processors with relatively cooler processors present in the platform at runtime. This could be a simple preventive measure to homogenize temperature among all processors using only the power consumption models and scheduling decisions. However, more sophisticated approaches that deal with thermal imbalance can be used. This aspect still remains a future perspective of this dissertation.

4.7 Concluding Remarks

In this chapter, we have presented a new dynamic power management technique, called the Assertive Dynamic Power Management (AsDPM) technique, for multiprocessor real-time systems. AsDPM is an admission control technique for real-time tasks which improves energy-efficiency of feasible task sets by deciding when exactly a ready task shall execute. Without this admission control, all ready tasks are executed as soon as there are enough computing resources (processors) available in the system, leading to poor possibilities of putting some processors in to power-efficient states. We have illustrated in this chapter that although, AsDPM is an online dynamic power management technique, its working principle can be used to determine static optimal architecture configurations (i.e., number of processors and their corresponding voltage-frequency level required to meet real-time constraints in worst-case with reduced energy consumption) for target application through simulations, thus allowing static as well as dynamic power and energy optimization. We have provided experiments with H.264 video decoder application to support our proposition. In the first step, we have demonstrated, through simulations, the use of the AsDPM technique for determining static optimal architecture configurations for H.264 video decoder application. In the second step, we have demonstrated the use of AsDPM as an online DPM technique which can further increase energy-efficiency of an already (statically) optimized architecture configuration for any given application. For H.264 video decoder application, average achieved energy gains range from 14% to 35% depending on different QoS requirements, while using AsDPM. Furthermore, we have briefly discussed the future perspectives of the AsDPM technique, especially from the point of view of energy-efficiency at memory subsystem level. We have briefly discussed heating issue in multiprocessor systems and possible thermal load-balancing while using AsDPM. The AsDPM technique is also integrated in a hybrid power and energy management scheme which is presented in chapter 6.

Chapter 5

Deterministic Stretch-to-Fit DVFS Technique

Contents

5.1 Dynamic Voltage and Frequency Scaling

Dynamic voltage reduction is one of the effective techniques, which aim at changing energy consumption profile of real-time embedded systems. This is because energy consumption of processors is a quadratic function of supply voltage of processors [48] (see section 2.2.1). However, reduction in supply voltage also requires reduction in operating frequency, which implies a trade-off between energy consumption and system's performance. Thanks to the significant evolution in processor-technology over the last few years, processors with variable voltages and frequencies are now available, thus allowing the design of highly flexible systems. Some examples of such processors are Marvell's XScale® technology-based embedded processors [72], Intel Speedstep [54], AMD PowerNow! [2], ARM [9], and Transmeta Crusoe [117]. In real-time systems that use variable voltage and frequency processors, choice of voltage and frequency has direct impact on processor's speed and consequently, on the ordering and the execution of tasks. Thus, scheduling techniques and voltage/frequency selection mechanisms are tightly coupled and should be addressed together to ensure *feasibility* of application tasks. In the last few years, *Dynamic Voltage and Frequency Scaling(DVFS)* techniques have emerged, which address the issue of energy-efficiency in real-time systems together with the scheduling. Real-time applications potentially exhibit variations in their

actual execution time and therefore, often finish earlier than their estimated worst-case execution time [10, 40]. Real-time DVFS techniques exploit these variations in *actual* workload in order to dynamically adjust voltage and frequency of processors to reduce their power and energy consumption. However, one of the challenges for these techniques is to preserve the *feasibility* of schedule and provide deadline guarantees.

Real-time DVFS techniques are broadly classified into *inter-task* and *intra-task* strategies [79]. Inter-task DVFS techniques redistribute dynamic slack time either to all ready tasks [51, 79] or to immediate priority single ready task only at *task boundaries*. That is, inter-task DVFS techniques make decisions related to slack reclamation only at scheduling events referring to the termination of a job. On the other hand, in intra-task strategies, available (dynamic) slack time is reallocated *inside* the same task. These techniques often require insertion points in application's code to measure its evolution over its execution time. Some major drawbacks of intra-task scaling methods are that they require excessive analysis, runtime tracking and modification of the task source code, which is not always feasible in reality [76, 79, 98, 121]. Furthermore, they normally result in large number of additional voltage and frequency switching points and most of them assume continuous voltage levels [96, 95]. In this chapter, we propose an *inter-task* dynamic voltage and frequency scaling technique for real-time multiprocessor systems, called *Deterministic Stretch-to-Fit (DSF)* technique. The DSF technique comprises three algorithms, namely, *Dynamic Slack Reclamation (DSR)* algorithm, *Online Speculative speed adjustment Mechanism (OSM)*, and *m-Tasks Extension (m-TE)* algorithm. The DSR algorithm is the principle slack reclamation algorithm of DSF that assigns dynamic slack, produced by a precedent task, to the appropriate priority next ready task that would execute on the same processor. While using DSR, dynamic slack is not shared with other processors in the system. Rather, slack is fully consumed on the same processor by the task, to which it is once attributed. Such greedy allocation of slack allows the DSR algorithm to have large slowdown factor for scaling voltage and frequency for a single task, which eventually results in improved energy savings. DSR works in conjunction with *global* scheduling algorithms on identical multiprocessor real-time systems. The OSM and the m-TE algorithms are extensions of the DSR algorithm. The OSM algorithm is an online, adaptive, and speculative speed adjustment mechanism, which anticipates early completion of tasks and performs *aggressive* slowdown on processor speed. Apart from saving more energy as compared to the stand-alone DSR algorithm, this speculative speed adjustment mechanism also helps to avoid radical changes in operating frequency and supply voltage, which results in reduced peak power consumption, which leads to an increase in battery life for portable embedded systems. The m-TE algorithm extends an already existing *One-Task Extension (OTE)* technique for single-processor systems onto multiprocessor systems. The DSF technique is implemented in conjunction with the EDF global scheduling algorithm.

The DSF technique is mainly intended for multiprocessor systems. Though, applying it on single-processor systems is also possible and in fact, rather trivial due to absence of migrating tasks. The DSF technique is generic in the sense that if a *feasible* schedule for a real-time target application exists under worst-case workload using (optimal or non-optimal) global scheduling algorithms, then the same schedule can be reproduced (using actual workload) with less power and energy consumption. Thus, DSF can work in conjunction with various scheduling algorithms. We illustrate that tasks meet their deadlines if the speed of processors is reduced, due to dynamic slack reclamation, according to the rules that we provide for the DSR algorithm. We achieve these objectives by exploiting only the *online* information of tasks. DSF is based on the principle of following the *canonical* execution of tasks at runtime –i.e., an offline or static optimal schedule in which all jobs

of tasks exhibit their worst-case execution time. A track of the execution of all tasks in static optimal schedule needs to be kept in order to follow it at runtime [10]. However, producing and keeping an entire canonical schedule offline is impractical in multiprocessor systems due to a priori unknown assignment of preemptive and migrating tasks to processors. Therefore, we propose a scheme to produce an *online canonical schedule* ahead of practical schedule, which *mimics* the canonical execution of tasks only for future m-tasks. This reduces scheduler's overhead at runtime as well as makes DSF an adaptive technique. The DSF technique can work for different processor technologies supporting discrete as well as continuous voltage and frequency scaling. Processors that are able to operate on (more or less) continuous voltage and frequency spectrum are becoming a reality [10]. Therefore, we provide our solution for systems that support both discrete and continuous dynamic voltage and frequency scaling.

In this chapter, we use the following notations from the literature on scheduling theory and low-power techniques.

Canonical schedule ***(S^{can})*** is a static optimal schedule in which each task instance presents its worst-case workload under a given scheduling algorithm.

Practical schedule ***(S^{pra})*** is an online schedule in which each task instance presents variations to its worst-case workload. These variations are bounded by best-case (B_i) and worst-case (C_i) execution times of that task.

Dynamic slack ***(ε)*** refers to the amount of time spared by a task due to its early completion. It is the difference between worst-case execution requirement and actual execution time exhibited by a task at runtime. It can be known only when a task terminates its running job.

Scaling factor ***(ϕ)*** refers to factor by which the worst-case execution time of a task is scaled (increased or decreased). It is calculated based on the amount of available slack (ε) on a processor and it is bounded by the task's deadline.

We assume that it is possible to vary supply voltage and operating frequency (also referred as a tuple (V_{dd}, F_{op}) hereafter) on every processor independently and over a continuous spectrum between defined lower and upper bounds. This assumption can be lifted for processors offering only discrete operating frequencies by adapting to the nearest possible operating frequency. The focus of this chapter in mainly on the online optimization of power/energy consumption techniques. Thus, we do not consider static optimization of (V_{dd}, F_{op})[1]. It is also assumed that statically specified *optimum* speed ϑ_{max} and corresponding optimum (V_{dd}, F_{op}) are known a priori. Speed of a processor at any time instant is referred as ϑ and minimum allowable processor speed is referred as ϑ_{min} such that ($\forall\vartheta$, $\vartheta_{min} \leq \vartheta \leq \vartheta_{max}$).

5.2 Related Work

In this section, we provide the reader a review of the stat-of-the-art on existing real-time DVFS techniques. Figure 5.1, which is adopted from [79] with slight modifications, gives a perspective of how the dynamic slack time is redistributed under different real-time DVFS

[1]The problem of static power optimization is briefly discussed in chapter 4

strategies. Figure 5.1 illustrates the evolution of processor's frequency during the execution of a job $T_{i,j}$ under different DVFS strategies. The worst-case execution time $C_{i,j}$ of $T_{i,j}$ is known a priori. However, job $T_{i,j}$ requires only $AET_{i,j}$ units of *actual execution time* to complete. Figure 5.1 (i) illustrates the case, in which $T_{i,j}$ executes with maximum frequency of processor, which is least effective from the point of view of energy consumption optimization. Figure 5.1 (ii) illustrates an *ideal stretch* strategy. An ideal stretch, however, is not feasible in practice because it is not possible to know a priori the exact amount of actual execution time of a task's job. Thus, it is impossible to scale processor's frequency with an ideal scaling factor. Nevertheless, an ideal stretch strategy provides a good benchmark to assess a posteriori performance of other DVFS strategies. Figure 5.1 (iii) and (iv) illustrate stochastic scheduling-based strategy and code instrumentation-based strategy, respectively. Both these strategies have opposite power consumption profiles. Indeed, the speed of processor increases with time in case of stochastic scheduling-based strategy (figure 5.1(iii)), while it decreases in case of code instrumentation-based strategy (figure 5.1(iv)) as the knowledge of remaining execution time becomes clearer. Figure 5.1 (v) illustrates the case in which dynamic slack time is distributed to all ready tasks. This technique results in small values of scaling factor for frequency, which eventually results in insignificant gains on energy consumption. Figure 5.1 (vi) illustrates the case in which processor's frequency is reduced such that a tasks's job finishes exactly at the same time instant, which was previewed worst-case workload at maximum frequency. The idea to stretch a task's execution time to its worst-case boundary, as illustrated in figure 5.1(vi), is used in single-processor systems. However, in multiprocessor systems, it is not trivial to apply simple stretching due to migrations and preemptions of jobs.

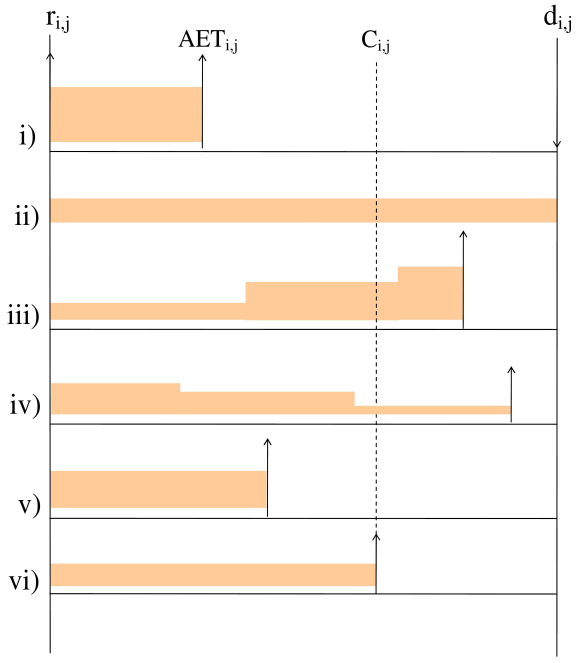


Figure 5.1: Dynamic slack redistribution of a task under various DVFS strategies.

Although real-time dynamic voltage and frequency scaling has become a major focus

by real-time research community in recent years, its proposition stems back as early as in 1994 when Weiser et al. first presented their work in [122]. Later on, numerous researchers have explored DVFS on single-processor systems such as [75, 98, 87, 10, 97, 122]. However, relatively few researchers have considered the problem of applying DVFS on multiprocessor platforms such as [50, 99, 124]. A recent survey by Chen et al. [25] summarizes the state-of-art on energy-efficient scheduling for real-time systems on DVFS platforms, including single-processor and multiprocessor systems.

For single processor systems, authors in [125] have proposed one of the earliest works, which presents an inter-task voltage scaling optimal algorithm to schedule a job set with specified arrival times and deadlines on a continuous voltage processor. The analysis and correctness of this algorithm is based on EDF scheduler on a single processor system. Authors in [66] later extend the algorithm proposed by [125] to compute optimal slowdown factors for the case of discrete voltage levels. Authors in [87, 10] have proposed inter-task DVFS techniques using single-processor optimal scheduling algorithms such as EDF and FPP. In [87], authors have presented a cycle-conserving approach in which cumulative resource utilization is recomputed at every scheduling event to rescale voltage and frequency appropriately. This approach, however, distributes dynamic slack time to all ready tasks which makes overall slowdown factor very small. Authors in [65] propose a power-aware scheduling technique using slack reclamation for systems with two voltage levels. Thus, limiting the number of scalable voltage levels to two. Authors in [75] have proposed and analyzed several techniques for dynamic speed adjustment of processors with slack reclamation. Authors in [91] propose an efficient heuristic technique to find the voltage schedule of an FP real-time system –i.e., different voltage settings at different time for job instances of aperiodic and periodic tasks.

In multiprocessor systems, the problem of energy minimization by dynamic slack reclamation and dynamic speed adjustment for globally scheduled systems and independent task model is considered in [129]. Authors in [129] consider dynamic slack sharing among multiple processors and reducing the speed globally on all processors. This work is extended in [128] to address the case of dependent tasks with AND/OR dependency constraints. Authors in [12] have proposed their solution for periodic hard real-time tasks on identical multiprocessors with DVS support when only partitioned scheduling is used. Authors in [12] adopt the EDF scheduling policy and investigate the joint effect of partitioning heuristics on the energy consumption and the feasibility. For systems with two-processors running at two different but fixed voltage levels, authors in [124] proposed a two-phase scheduling scheme that minimizes energy consumption under the time constraints by choosing different scheduling options determined at compile time. Authors in [10] have proposed an inter-task voltage scheduling solution for hard real-time systems comprised of a static (offline) solution to compute the optimal speed beforehand and an online solution based on dynamic speed reduction mechanism. Authors in [127] highlight that processor energy optimization solutions are not necessarily efficient from the perspective of whole system. They provide their solution for periodic and sporadic task models with a perspective of system-wide energy-efficiency.

Apart from *dynamic* voltage and frequency scaling techniques for real-time multiprocessor systems, some of the existing research work focuses on real-time *static* voltage and frequency scaling as well [94, 46, 45]. The thesis for such work is based on the fact that changing voltage and frequency takes some time to ensure system synchronization due to physical limitations. For instance, Intel Pentium M [49] processors require $10 - 15\mu$s per voltage and frequency scaling. It is possible to ignore the overhead in some cases. However, next-generation real-time systems are more rapid and controlled in faster control loops. For

such systems, frequent voltage and frequency scaling incurs significant overhead. Furthermore, the optimal real-time scheduling algorithms for multiprocessors cause more frequent context switches than other algorithms. Thus, static voltage and frequency scaling is a good solution for these systems. One such solution is provided in [46]. Authors in [46] have highlighted that, in currently available practical systems, optimal real-time static voltage and frequency scaling on multiprocessors is NP-hard partition problem since selectable processor frequency is discontinuous. Although static optimizations for voltage and frequency are desirable, solutions based on only SVFS techniques are not good enough for systems that are likely to vary their configurations without system standstill. Moreover, systems with aperiodic tasks would become sub-optimized online if the tasks' arrival frequency varies. DVFS techniques are known to reduce dynamic power consumption (see section 2.2.1). However, apart from latency issue related to changing voltage and frequency, DVFS techniques also cause increased leakage energy drain by lengthening the interval over which a computation is carried out. As a solution, authors in [59] have proposed a leakage-aware DVFS technique for real-time systems in which, processor slowdown factors are computed based on the *critical speed* for energy minimization. They have shown up to 5% gains using critical speed slowdown over a leakage oblivious DVFS solution.

The DSF technique is intended for periodic and independent tasks, which are scheduled over multiprocessor real-time systems. DSF is based on the idea of stretching the execution time of a job, which consumes dynamic slack, to its worst-case boundary as illustrated in figure 5.1(vi). This stretching permits a hard real-time application to benefit from the same timeliness guarantees as provided by worst-case schedulability analysis since the aggregate utilization of task set does not change due to processors' frequency scaling.

5.3 Deterministic Stretch-to-Fit Technique

In this section, we present the algorithms proposed in the DSF technique –i.e., the DSR algorithm, the OSM algorithm, and the m-TE algorithm.

5.3.1 Dynamic Slack Reclamation (DSR) Algorithm

Timeliness guarantees for real-time applications are provided through statically performed worst-case schedulability analysis. This, however, is a conservative analysis because at runtime, real-time tasks can exhibit large variations in their actual execution time and thereby produce dynamic slack due to their early completion [39]. The DSR algorithm is based on detecting early completion of tasks. DSR determines the amount of dynamic slack (if any) by comparing the worst-case execution requirement of a task's job with its actual execution requirements. Since it is not possible to know a priori the exact amount of actual execution requirement of a running task until it terminates, therefore, DSR computes the amount of dynamic slack at task boundaries only.

The DSR algorithm does not not share dynamic slack with other processors in the system. Rather, the slack is fully consumed on the same processor by a subsequent job, which is assigned to that processor. For instance, if a terminating job, let us say $T_{i-1,j}$, produces positive dynamic slack time ε on a processor π_k then the DSR algorithm, based on the amount of available slack, determines a scaling factor ϕ for subsequently assigned appropriate priority job, let us say $T_{i,j}$, on π_k. Scaling factor is calculated such that the *consumer* task (i.e., task that consumes ε) finishes its *slowed down* execution by at most the same time instant of its worst-case boundary as defined in the canonical schedule (S^{can}). An issue of concern in calculating ϕ is the knowledge of entire canonical schedule of tasks.

In multiprocessor systems, unlike the single-processor systems, producing and keeping a priori an entire canonical schedule and then following it online is clearly impractical in the absence of tie-breaking rules and preemptive and migrating task models. This is due to the fact that tasks migrate in non-partitioned scheduling. Moreover, different jobs of the same task can be assigned to different processors in two successive simulation traces even if their priorities are the same. As a solution, we produce an *online* canonical schedule of tasks ahead of practical schedule that mimics the canonical execution of tasks at runtime. We present the criterion for producing online canonical schedule in section 5.3.2. At this stage, we assume that the canonical schedule for given task set is known beforehand.

Algorithm 6 Dynamic Slack Reclamation

```
1:  obtain   S^can
2:  sort     ready tasks queue w.r.t. preemptive EDF priority order
3:  assign   m highest priority tasks on m processors
4:  set      ε ← 0;   φ ← 1.0;   ϑ ← ϑ_max
5:  for each scheduling event do
6:    if    scheduling event=termination then
7:          ε ← C_{i−1,j} − AET_{i−1,j}
8:      if   ε > 0 then
9:           compute available time t_av for T_{i,j} at ϑ      (t_av=C_{i,j} + ε)
10:          compute required time C_{i,j} by T_{i,j} at ϑ     (C^ϑ_{i,j})
11:          φ ← t_av/C^ϑ_{i,j}
12:          update ϑ w.r.t. φ
13:       if   ϑ < ϑ_min then
14:            ϑ ← ϑ_min
15:       execute T_{i,j} at ϑ
16:     else if  ε = 0 then
17:          ϑ ← ϑ_max
18:   else if  scheduling event=release then
19:     if    released task has higher priority then
20:          preempt least priority task
21:          ϑ ← ϑ_max
```

The DSR algorithm exploits the fact that different scheduling events have different impact on an application's schedule. For instance, a terminating job may produce dynamic slack, but it does not increase concurrent utilization of the platform and therefore, can only update the priority order of remaining ready tasks. A release event, on the other hand, increases the concurrent platform utilization and may cause preemptions as well. This difference in the impact of scheduling events is exploited by DSR. Pseudo code for DSR is provided in algorithm 6.

In algorithm 6, for a feasible task set, the S^{can} is assumed to be known a priori and all tasks are sorted according to their priority order under the preemptive EDF scheduling algorithm (lines $1-2$). Highest priority m-tasks are assigned to m-processors of platform (line 3). Parameters like dynamic slack (ε), scaling factor (ϕ), and initial speed (ϑ) for all processors are set to 0, 1.0, and ϑ_{max}, respectively (lines $1-4$). Once the system is initialized, DSR updates ϑ only at scheduling events. Whenever a scheduling event arrives, DSR makes a distinction between the release and the termination events (line 6 and line 18).

If a termination event is detected on a processor π_k, DSR examines whether the completing job (let us say $T_{i-1,j}$) has generated any positive dynamic slack (line 7). If dynamic slack is available (i.e., $\varepsilon > 0$) then the execution of subsequent appropriate priority job (let us say $T_{i,j}$) can be *slowed down*. To do so, the algorithm computes additional available time t_{av} on π_k at current processor speed ϑ. The worst-case execution time $C^{\vartheta}_{i,j}$ of $T_{i,j}$ is estimated at ϑ that *would have been* required if π_k continues to run at current speed ϑ (lines $8-10$). Once these two values are known, the scaling factor for $T_{i,j}$ is computed and ϑ is updated on π_k accordingly (lines $11-12$). It is possible that the precedent job $T_{i-1,j}$ has produced enough dynamic slack to reduce ϑ below the allowable minimum processor speed ϑ_{min}. To avoid such situation, ϑ is bounded by ϑ_{min} in DSR (lines $13-15$). If there is no positive slack generated by $T_{i-1,j}$, processor π_k continues to execute with maximum speed ϑ_{max} (lines $16-17$).

If, on the other hand, a release event is detected on processor π_k, DSR examines if the released job has its priority higher than the currently running m-tasks. If so, least priority running job is preempted in order to run higher priority job and the speed of processor is set to maximum (lines $18-21$). Note that when a higher priority task is released, the DSR algorithm increases the speed of processor to statically determined maximum (ϑ_{max}) in order to respect the worst-case schedulability of released job. This is due to the fact that actual execution time of preempting task is unknown a priori. A speculative speed adjustment mechanism can be helpful in determining a *probabilistic* actual execution time that would allow to keep the speed of processor less than ϑ_{max}. Note that if the *preempted* job was executing with a speed other than ϑ_{max} then its remaining execution time is always restored w.r.t. maximum speed -i.e., ϑ_{max} so that, upon resuming its execution, the preempted job can always meet its deadline. In the following, we provide a simple example to elaborate the working of DSR. In this example, we provide an analytical view at first and then support it with a simulation trace.

Example 5.1: Let us consider two real-time tasks, namely T_1 and T_2, which are scheduled under the EDF scheduling algorithm. Figure 5.2 illustrates that T_1 has higher priority over T_2 due to smaller deadline. Let us consider that the job $T_{1,j}$ of T_1 finishes its execution earlier than its worst-case execution requirement ($C_{1,1}$) for its first job $T_{1,1}$ as shown in inset (a). In the canonical schedule S^{can} –i.e., if job $T_{1,1}$ would have finished with its worst-case execution time, the termination of $T_{1,1}$ was estimated at time instant $t^{can}_{1,1}$. However, $T_{1,1}$ terminates at time instant $t^{pra}_{1,1}$ after consuming $AET_{1,1}$ time units and produces ε units of dynamic slack. Similarly, job $T_{2,1}$ of T_2 was expected to start its execution at time instant $t^{can}_{1,1}$ and finish by $t^{can}_{2,1}$ as illustrated in inset (b). However, due to the early completion of $T_{1,1}$, now $T_{2,1}$ has t_{av} time units available (rather than $C_{2,1}$ time units under worst-case) to finish its execution as expected in its S^{can}. Since $AET_{2,1}$ is not known a priori, therefore, $C_{2,1}$ is considered as the execution requirement for $T_{2,1}$. Based on the knowledge of available slack ε and task boundary in S^{can} (i.e., time instant $t^{can}_{2,1}$), execution of $T_{2,1}$ is slowed down by a factor of ϕ as shown in inset (c).

We simulate the same example using STORM simulator to validate the concept. Let us consider that the quadruplet values for T_1 and T_2 are $(0, 6, 8, 8)$ and $(0, 5, 20, 20)$, respectively. Let us consider that T_1 has a best-case execution time of 3 time units and often finishes earlier than its worst-case, whereas T_2 always executes with its worst-case execution time. Figure 5.3(a) illustrates the canonical schedule of T_1 and T_2, in which both tasks always consume their respective C_i time units at runtime. On the other hand, figure 5.3(b) illustrates a practical schedule of the same tasks, in which T_1 always finishes with its best-case execution time –i.e., 3 time units, and produces a dynamic slack of 3 time units ($\varepsilon = 3$). In figure 5.3(b) and figure 5.4, it is shown that task T_2 consumes ε time units

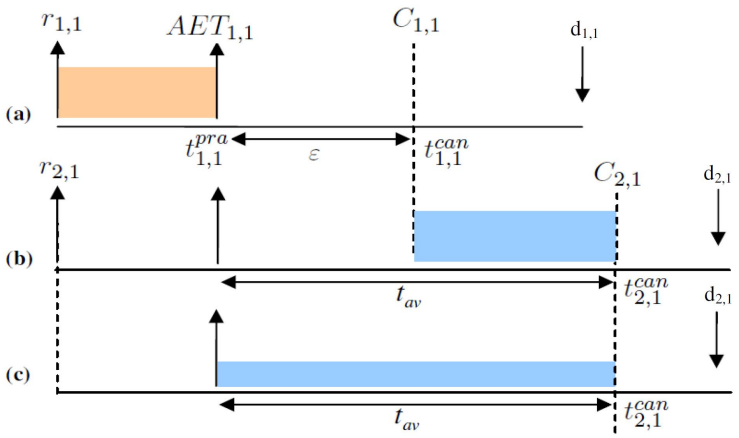


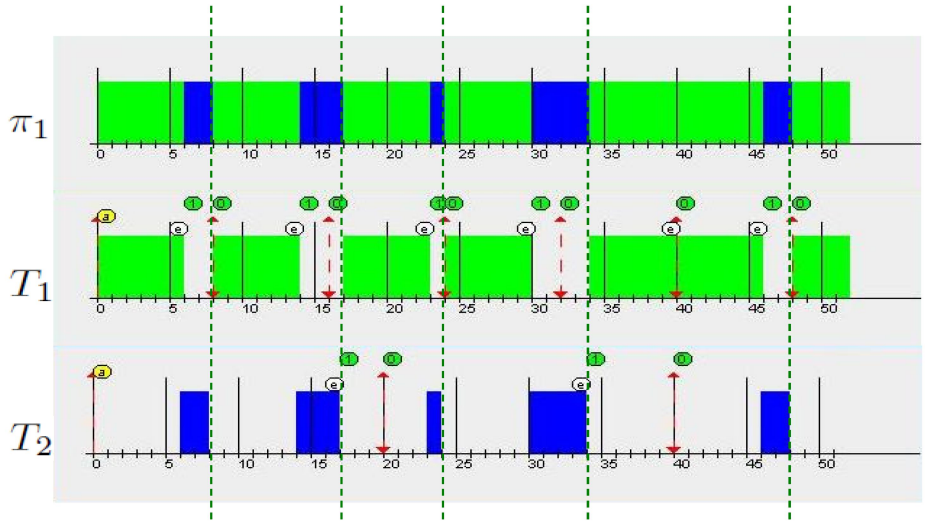
Figure 5.2: Slack reclamation using the DSR algorithm.

of dynamic slack to elongate its own execution time. However, the elongated execution time always coincides with the boundaries of T_2 in its canonical schedule at runtime. For instance, when first job $T_{1,1}$ of T_1 finishes at time instant 3 in S^{pra} and generates 3 time units of slack, the first job $T_{2,1}$ of T_2 starts executing early at time instant 3 but still finishes at time instant 8 which coincides with its termination instant in S^{can}. Vertical dotted lines highlight termination instants of T_2 in both S^{can} and S^{pra}.

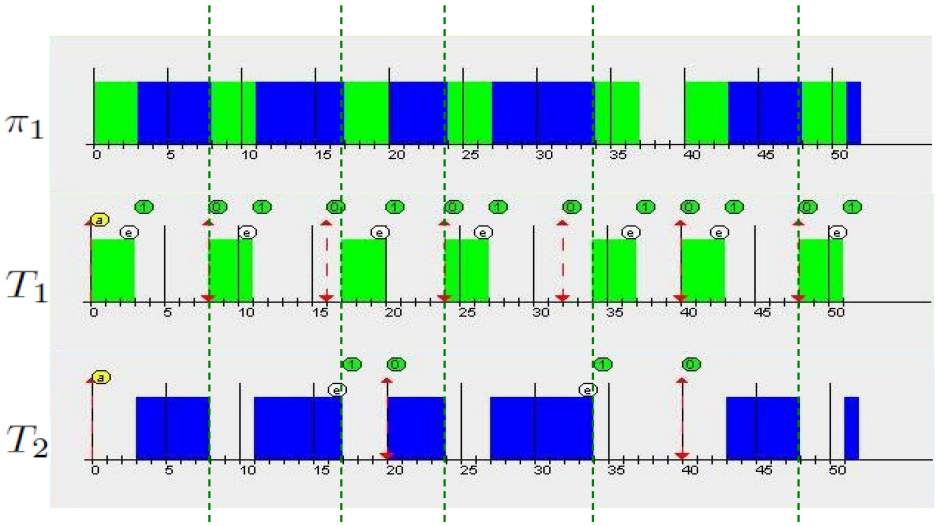
Moreover, in figure 5.4, it can be noticed in S^{pra} that at simulation time instant 37, dynamic slack is available but not exploited. This is due to the fact that there is no subsequent ready task available in ReTQ. This slack time could have been exploited using the m-Task Extension algorithm, which scales down the processor frequency when there are only m tasks left in ReTQ. Section 5.3.4 elaborates the m-TE technique.

5.3.2 Online Canonical Schedule

For single-processor systems, it is possible to statically construct an entire canonical schedule that does not change at runtime over successive hyper-periods if all tasks execute with their worst-case workload and tie-breaking rules are applied. Moreover, a canonical schedule for single-processor systems is not very complex to store offline and then follow at runtime. Unfortunately, for multiprocessor systems, it is not trivial to construct an offline canonical schedule. In multiprocessor systems, unlike single-processor systems, producing and keeping a priori an entire canonical schedule offline and then following it at runtime is impractical due to the fact that tasks migrate to other processors in non-partitioned scheduling. Even if the priorities of tasks are similar over multiple hyper-periods and tie-breaking rules in place, tasks can be allocated to different processors by the scheduler over successive hyper-periods. For instance, at a given scheduling event, if two processors are available to execute two fully migrating equal priority tasks, then the scheduler can assign tasks to processors in any order. Thus, a practical schedule can result in different ordering of tasks than statically constructed canonical schedule. Nonetheless, storage of entire canonical schedule beforehand requires additional memory space and additional workload for scheduler to retrieve stored schedule at runtime. Thus, it becomes impractical in multiprocessor systems with fully preemptive and migrating tasks to follow a statically constructed canonical schedule.



(a) Canonical schedule of tasks where all tasks execute with worst-case execution time.



(b) Practical schedule of tasks where T_1 finishes earlier than its WCET and T_1 exploits dynamic slack to elongate its WCET at runtime.

Figure 5.3: Simulation traces of example task set on a single processor. a) Canonical schedule of tasks where all tasks execute with worst-case execution time. b) Practical schedule of tasks where T_1 finishes earlier than its WCET and T_1 exploits dynamic slack to elongate its WCET at runtime.

As a solution, we propose to construct an *online* (reduced version of) canonical schedule ahead of practical schedule in multiprocessor systems. This online canonical schedule is constructed only for the future m-tasks that are present in ReTQ. For illustration purpose, we can state that a multiprocessor global scheduling algorithm maintains, at runtime, at least three types of sorted task queues as presented in section 4.3 of chapter 4, that is:

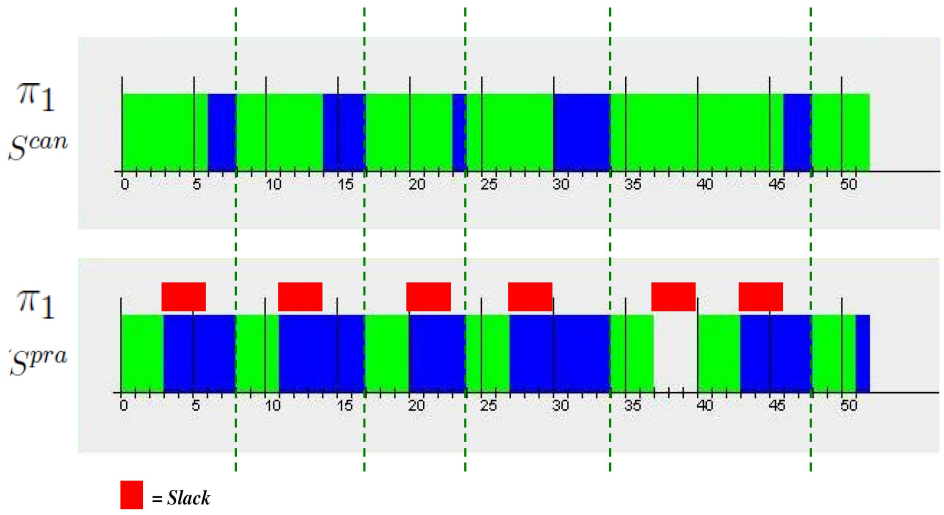


Figure 5.4: Task T_2 consumes ε to elongate its execution up to its termination instant in canonical schedule.

running tasks' queue (RuTQ), ready tasks' queue (ReTQ) and a general purpose tasks' queue (TQ) for all non-ready and non-running tasks as illustrated in figure 5.5.

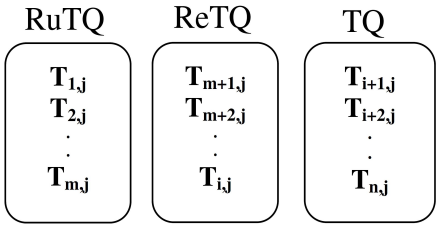


Figure 5.5: Task queues managed by a global scheduler at runtime.

Let us consider a multiprocessor system composed of m processors having n tasks to be scheduled under the EDF global scheduling algorithm such that $n \geq m$. At any time instant t, at most m tasks can be running on m-processors (and therefore present in RuTQ). Since it is not possible to know a priori the exact amount of actual execution time ($AET_{i,j}$) of these running tasks before they complete, therefore, we make a conservative assumption that all tasks in RuTQ consume their worst-case execution time before their completion. Rest of $n - m$ tasks should be present in either ReTQ or TQ. Figure 5.6 illustrates that jobs $T_{1,1}$, $T_{2,1}$, and $T_{3,1}$, being the highest priority jobs, run over processors π_1, π_2, and π_3 respectively. Let us construct an online canonical schedule at time instant t as shown in figure 5.6. From the sorted ReTQ and TQ queues, the next highest priority task, lets say T_{next}, can be identified using equation 5.1 (also see figure 5.5 as well). Here, $T_{m+1,j}$ and $T_{i+1,j}$ represent the highest priority tasks in ReTQ and TQ, respectively. Since all running tasks are supposed to consume their worst-case execution requirement, therefore, the earliest possible time instant $t_{earliest}$ at which any of the m processors will become available for next

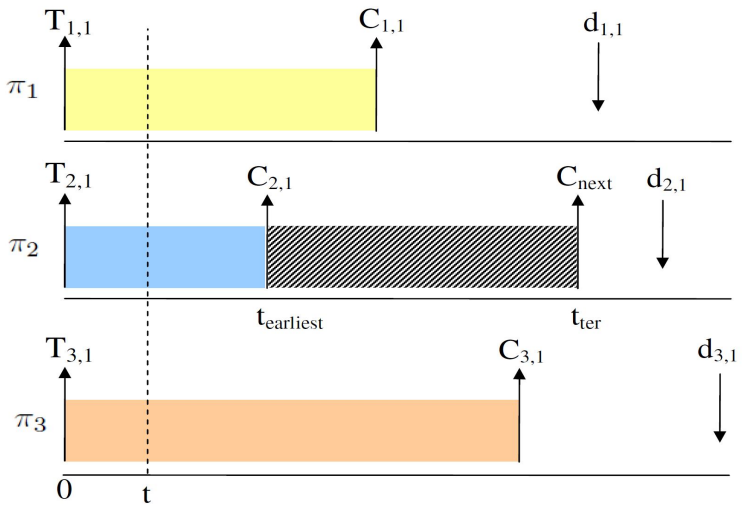


Figure 5.6: Construction of online canonical schedule ahead of practical schedule for m-tasks.

task allocation can be identified using equation 5.2[2]. For instance, in figure 5.6, $t_{earliest}$ arrives on π_2. In S^{can}, T_{next} will start executing on π_2 at time instant $t_{earliest}$. Once T_{next} and $t_{earliest}$ are identified, the time instant t_{ter}, at which T_{next} will finish, is identified by equation 5.3. It is sufficient to determine T_{next} for at most m processors of the platform at any scheduling event to construct an online canonical schedule ahead of practical schedule. Doing so, requires storage capacity for the canonical schedule of only m tasks.

$$T_{next} = max\_priority\left[\left(T_{m+1,j}\right), \left(T_{i+1,j}\right)\right] \tag{5.1}$$

$$t_{earliest} = t + min_{i=1}^{m}\left[C_{i,j}\right] \tag{5.2}$$

$$t_{ter} = t_{earliest} + C_{next} \tag{5.3}$$

Canonical schedule constructed in an online fashion is adaptive to the changes in task allocation on different processors over different hyper-periods. Note that, at any time instant t referring to a scheduling event, the online canonical schedule is only a *virtual projection* of how the schedule will evolve over time. Moreover, this evolution is limited by the number of tasks present in ReTQ. If another scheduling event occurs before the projected termination of running jobs then the online canonical schedule also updates its projection. This ensures that the practical schedule always follows a correct canonical schedule at runtime.

A real-time task set, which is feasible, remains feasible as far as its aggregate utilization remains the same, for which static schedulability analysis holds. The idea of DSR is based on the fact that no task shall execute beyond the time instant that corresponds to its worst-case completion in online canonical schedule (S^{can}). Moreover, under the DSR algorithm,

[2]Note that only the remaining execution time of $C_{i,j}$ is considered at time instant t while using equation 5.2.

the aggregate utilization of task set remains constant because whatever slack is produced, due to early completion of precedent tasks, is consumed by subsequent tasks and no task executes beyond its worst-case completion time instant in online canonical schedule (S^{can}).

5.3.3 Online Speculative speed adjustment Mechanism (OSM)

In section 5.3.1, we have mentioned that when a higher priority task is released and it preempts an already running lower priority task, the DSR algorithm increases the speed of processor to statically determined maximum (ϑ_{max}) in order to respect worst-case timing constraints of released job. This is due to the fact that actual execution time of preempting task remains unknown until it finishes its execution. Thus, processor speed cannot be kept lower than ϑ_{max}. If processor speed is reduced for a newly released hard real-time job, it can cause the deadline miss. In real-time systems, not all the tasks are hard real-time. Often there are some soft real-time tasks which coexist with hard real-time tasks. In this section, we extend the DSR algorithm with an online *speculation* mechanism for soft real-time tasks. Whenever, under maximum speed (ϑ_{max}), tasks (both soft and hard real-time tasks) exhibit large variations in their execution time, starting a task with the assumption that it will execute with its worst-case workload can be too conservative. For hard real-time tasks, speculation about their WCET is not possible. However, an online speculation at release instant of a soft real-time job can be helpful in determining its *probabilistic* actual execution time, which could help in maintaining the processor speed lower than maximum. We propose an online speculative speed adjustment mechanism based on the *average execution behavior* of soft real-time tasks. With the help of runtime profiling, it is possible to maintain a history of actual execution time exhibited by each task fr all its past jobs. This actual execution time is then averaged to achieve a *speculative* execution time for future jobs. Use of speculative workload helps avoiding a radical change in processor speed and improves energy savings. However, this speculative move might shift a task's worst-case completion time to a point later than the one in S^{can} under an actually high workload. If this pessimistic scenario turns out to be true, processor's speed should be increased later (for instance, when a job reaches its termination time instant in S^{can}) to guarantee feasibility of future jobs. Upon completion of every job, the speculation on next job is updated using average actual execution time.

Algorithm 7 presents the pseudo-code for the OSM algorithm. It can be noticed that every time a real-time job finishes, the actual execution time ($AET_{i,j}$) for its most recent job is used to update an *average* actual execution time ($AvAET_{i,j+1}$) for the next job $T_{i,j+1}$ (lines $1-4$) using all previous job instances that have occurred up to current time instant. Once updated, this average execution time is treated as a probable worst-case execution time $C_{i,j+1}$ for succeeding job of the same task (line 5). The DSR algorithm is then called to evaluate scaling of processor speed, if possible, in a normal fashion (line 6). In the presence of speculation mechanism, algorithm 6 does not execute a soft real-time task with ϑ_{max} upon its release. Rather, it uses $C_{i,j+1}$ of all jobs to figure out an appropriate value for ϑ and therefore, reduces peak power and overall energy consumption.

Note that, currently, the average actual execution time ($AvAET_{i,j+1}$) for the next job $T_{i,j+1}$ of each task is updated using *all* previous job instances that have occurred up to current time. This approach may not be very efficient and lead to inaccurate speculation because certain tasks may have different behavior (different variations in their AET) during different phases of application's execution. For instance, a task belonging to image processing application may have small, medium, or large variations in its AET while processing different image frames. Thus, more sophisticated mechanism to update $AvAET_{i,j+1}$ shall be used for OSM.

Algorithm 7 Online Speculation Mechanism

1: **for** each scheduling event **do**
2: **if** scheduling event=termination **then**
3: obtain $AET_{i,j}$ of terminating task's job $T_{i,j}$
4: $AvAET_{i,j+1} \leftarrow \left(\sum_{u=1}^{j-1} (AvAET)_u + AET_{i,j}\right)/j$
5: $C_{i,j+1} \leftarrow AvAET_{i,j+1}$
6: call the DSR algorithm

5.3.4 *m*-Tasks Extension Technique (m-TE)

This technique actually extends the *One-Task Extension* technique from single-processor systems onto multiprocessor systems. As demonstrated in [10, 97] that in a single processor system, one can further slow down the speed of processor when there is only one task left in the ready task's queue (ReTQ) and its worst-case completion time at current speed ϑ does not extend beyond the next scheduling event (next arrival or closest deadline of any task). One-task extension technique can be used in conjunction with any scheduling policy. To demonstrate how m-tasks extension technique works, let us start from a single-processor system. Suppose that $T_{i,j}$ is the last remaining ready job in ReTQ at time instant t and the earliest possible release instant of any other task present in TQ is $t_{earliest}$ such that $t_{earliest} > t$. If $(Y = t_{earliest} - (t + C_{i,j}) > 0)$ at current processor speed ϑ then the execution of $T_{i,j}$ can be further slowed down to consume additional Y time units, implied that the lower bound on speed ϑ_{min} is respected. One task extension technique can be extended to multiprocessor systems as shown in algorithm 8 using the same criterion. If there are at most m tasks left in ready task's queue (lines 1-2) then the available execution time for each ready task can be extended up to the earliest time instant corresponding to a scheduling event -i.e., either deadline of task or earliest next release r_{next} (lines 4-6). The earliest next release instant r_{next} refers to the arrival time instant of the highest priority job from TQ. Scaling factor ϕ is then updated using newly updated available time t_{av} and $C^{\vartheta}_{i,j}$. Processors' speed is updated using ϕ (line 8) as in algorithm 6.

Algorithm 8 m-Tasks Extension Technique

1: **for** each scheduling event **do**
2: **if** tasks in ReTQ $\leq m$ **then**
3: obtain earliest release instant r_{next} from TQ
4: **for** each task in ReTQ **do**
5: **if** $C_{i,j} \leq r_{next}$ **then**
6: $t_{av} \leftarrow \min(d_{i,j}, r_{next})$
7: $\phi \leftarrow t_{av}/C^{\vartheta}_{i,j}$
8: update ϑ w.r.t. ϕ

5.4 Experiments

In this section, we provide simulation-based evaluation of DSF. These simulations are carried out with mainly two objectives in mind. First is to validate that applying DSF does not effect the schedulability of real-time tasks and second is to evaluate the gains on overall

energy savings of the system.

5.4.1 Setup

We evaluate the performance of DSF using STORM simulator [108]. We consider the same system model –i.e., task model, processing platform, and power and energy models, as presented in chapter 2. One of the difference w.r.t. system model presented in chapter 2 is that, we assume that it is possible to vary supply voltage and operating frequency (V_{dd}, F_{op}) on every processor independently and over a continuous spectrum between defined lower and upper bounds. This assumption can be lifted for processors that offer only a finite number of discrete operating frequencies. As suggested in [22], if the desired optimal speed (corresponding to the achievable scalability of voltage/frequency) is not available on a processor, it has to be approximated with one of the existing values. To prevent any deadline miss, the processor speed should be set equal to the closest discrete level higher than the optimal speed. This solution, however, may cause a waste of computational capacity and, consequently, of energy, especially when the choice for available operating frequencies is limited.

The EDF scheduling algorithm is used in the evaluation of DSF.

5.4.2 Target Application

The H.264 video decoder application is taken as main use case target application for evaluating DSF. The main steps of the H.264 decoding process are depicted in figure 4.4 of section 4.5.1. In simulations related to the DSF technique, we use both versions (i.e., slices and pipeline versions) of H.264 video decoder application as presented in chapter 4.

5.4.3 Simulation Results

We provide simulation results for both versions of H.264 video decoder application using different throughput requirements. Recall that throughput requirement is a user-specified parameter, measured as frames per second (fps), which determines different levels of Quality of Service (QoS) for the H.264 application. The number of processors corresponding to different QoS (fps) requirement at maximum operating voltage and frequency level are already determined (see table 5.1) using AsDPM presented in section 4.4 (see chapter 4). Performance of the DSF technique is analyzed based on energy consumption compared to non-optimized case while using DSR lone, both DSR and OSM together, and using all three algorithms together –i.e., DSR, OSM, and m-TE. Furthermore, we have compared our results with two existing DVFS algorithms. Details are presented in section 5.4.3.4.

5.4.3.1 Simulation results of H.264 slices version

We simulate the task model presented in table 4.1 under the simulation settings provided in table 5.1. The bcet/wcet ratio is varied between 50% and 100% of wcet of tasks such that AET of all tasks has a uniform probability distribution function as suggested in [10]. For different frame rates, we obtain results as illustrated in figure 5.7. These results represent the average energy consumption over three complete frames (i.e., hyper-periods). In figure 5.7, it can be noticed that energy gains of the DSF technique, while using all its algorithms in different combinations, result from 12% to 35% as compared to the non-optimizaed energy consumption under EDF schedule. In the best-case, energy gains reach upto 45%, which reduces energy consumed per frame to 1.5J/frame as compared to 2.45J/frame in

Table 5.1: Simulation settings for H.264 video decoder slices version

Frame rate (fps)	No. of tasks (n)	No. of processors (m)	bcet/wcet ratio
8.33	7	3	50% – 100%
10.0	7	3	50% – 100%
11.11	7	4	50% – 100%
15.15	7	4	50% – 100%
17.24	7	4	50% – 100%
20.83	7	6	50% – 100%
22.27	7	6	50% – 100%

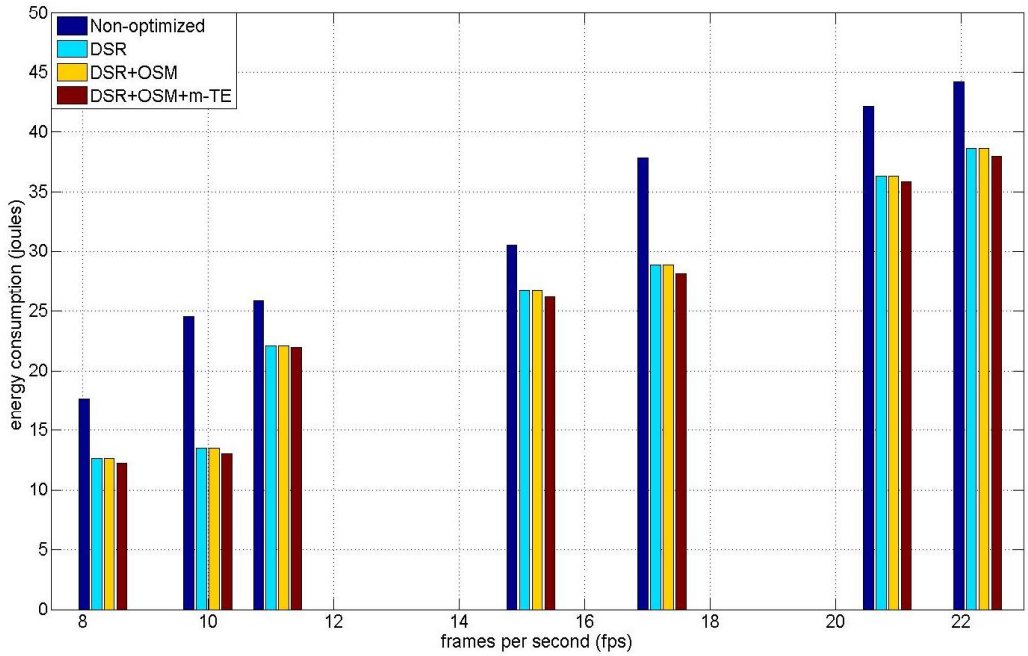


Figure 5.7: Simulation results of H.264 slices version.

non-optimized case. Moreover, it can be noticed that when the DSR algorithm is applied alongwith its extension algorithms –i.e., OSM and m-TE, gains on energy savings can be a little bit further improved. In these results, however, there is no difference in energy savings when DSR is used with OSM only. The reason for no change in energy in this case is that tasks presented in table 4.1 are all considered as hard real-time and therefore, no speculation is performed on the actual execution time of tasks during these simulations. We have illustrated the effectiveness of OSM in section 5.4.3.3.

5.4.3.2 Simulation results of H.264 pipeline version

Similar to section 5.4.3.1, we perform simulations for H.264 pipeline version of tasks presented in table 4.2. Simulation settings for pipeline version are given in table 5.2. Simulation

results obtained in this case are presented in figure 5.8. In figure 5.8, it can be noticed that energy gains of DSF, while using all its algorithms in different combinations, result from 5% to 35% as compared to the non-optimizaed energy consumption under EDF schedule. In the best-case, the energy consumed per frame is reduced to 0.45J/frame compared to 0.73J/frame in non-optimized case. When the DSR algorithm is applied along with its extension algorithms –i.e., OSM and m-TE, it works almost the same way as in section 5.4.3.1.

Table 5.2: Simulation settings for H.264 video decoder pipeline version

Frame rate (fps)	No. of tasks (n)	No. of processors (m)	bcet/wcet ratio
10	7	1	50% – 100%
12	7	1	50% – 100%
15	7	2	50% – 100%
20	7	2	50% – 100%
25	7	2	50% – 100%
32	7	2	50% – 100%

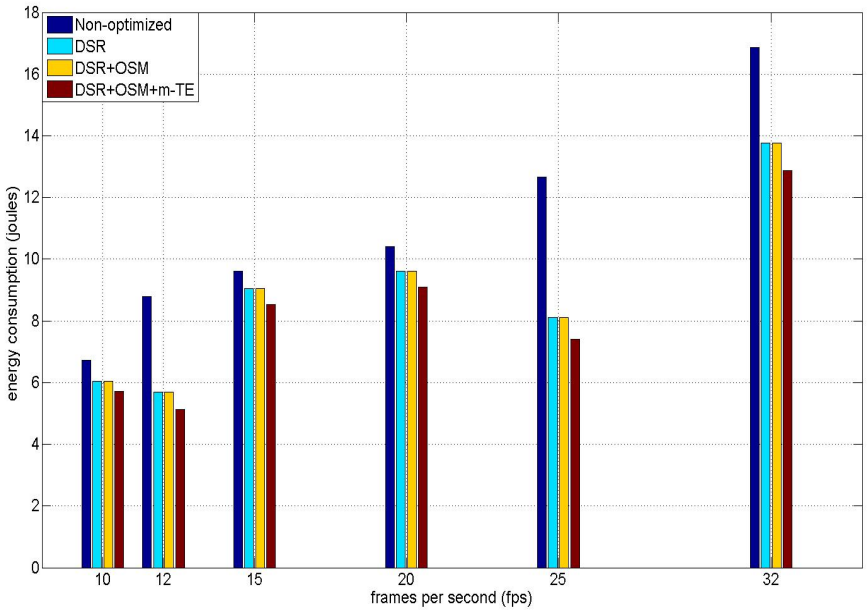


Figure 5.8: Simulation results of H.264 pipeline version.

5.4.3.3 Simulation results of H.264 pipeline version: Effect of OSM

In the results presented so far, we consider that all tasks of H.264 video decoder application are hard real-time tasks and therefore, the online speculation mechanism can not speculate

an average actual execution time for tasks. In order to demonstrate the effectiveness of OSM, we assume that it is possible to speculate on the AET of task T_3 (RE−1), T_4 (RE−2), and T_5 (RE-F). Figure 5.9 illustrates that speculating on the AET of task slightly improves energy savings. For speculative AET for large number of tasks, energy gains could become significant.

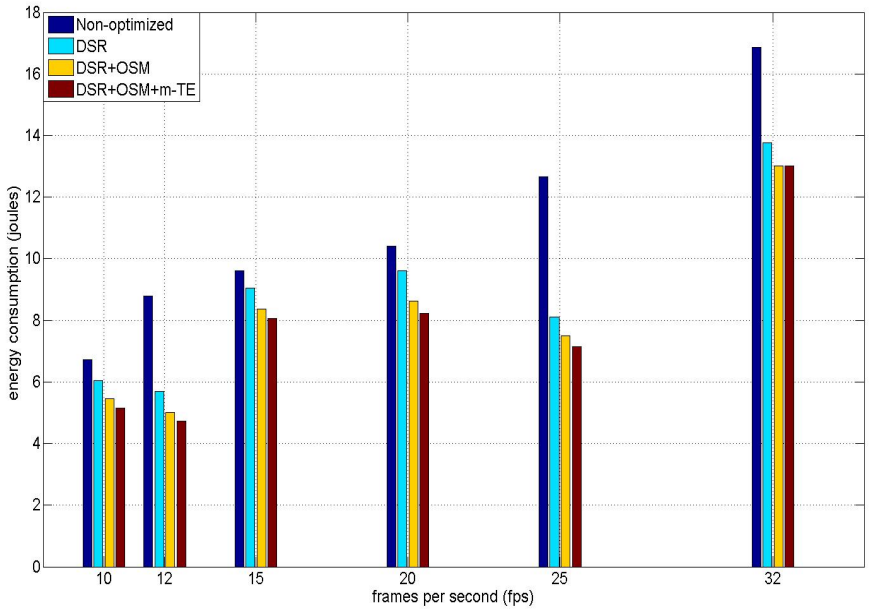


Figure 5.9: Simulation results of H.264 pipeline version illustrating the effectiveness of OSM.

5.4.3.4 Comparative analysis

In this section, we compare simulation results obtained for both versions of H.264 video decoder under non-optimized EDF scheduling technique, the ccEDF (DVFS) technique [87], the RBSS (DVFS) technique [88], and the DSF (DVFS) technique. A brief description of the ccEDF and the RBSS DVFS techniques is provided in the following.

ccEDF (Cycle-conserving EDF) is a classical DVFS technique presented in [87]. In this technique, dynamic slack time is redistributed globally to all ready tasks. The ccEDF technique considers actual execution time (AET) of all terminated tasks until their next release and worst-case execution time of all ready and running tasks to determine concurrent resource utilization of target application. Based on these values, ccEDF then calculates required scaling factor for speed. Once scaling factor is determined, frequency and voltage are scaled globally on all processors.

RBSS (Resource Based Slack Sharing Algorithm) is coupled to a job-level dynamic-priority scheduling policy, called *Enhanced Least Laxity First (ELLF)*, allowing task preemption and migration for a better resource utilization. In this method, the slack

is bound to the originator resource meaning that a slack produced on a processor is distributed to the next task allocated on it. To ensure a real-time execution, the slack reclaimed is upper bounded by the task's laxity. This slack decrements with the time if no other task is allocated on this processor. Real-time periodic application tasks start execution on all the processors at maximum frequency (or at an offline computed frequency allowing to meet the application deadline considering task WCETs) and changing to a more interesting voltage and frequency level when cumulating enough slack on a given processor. Further details on RBSS algorithm can be found in [88].

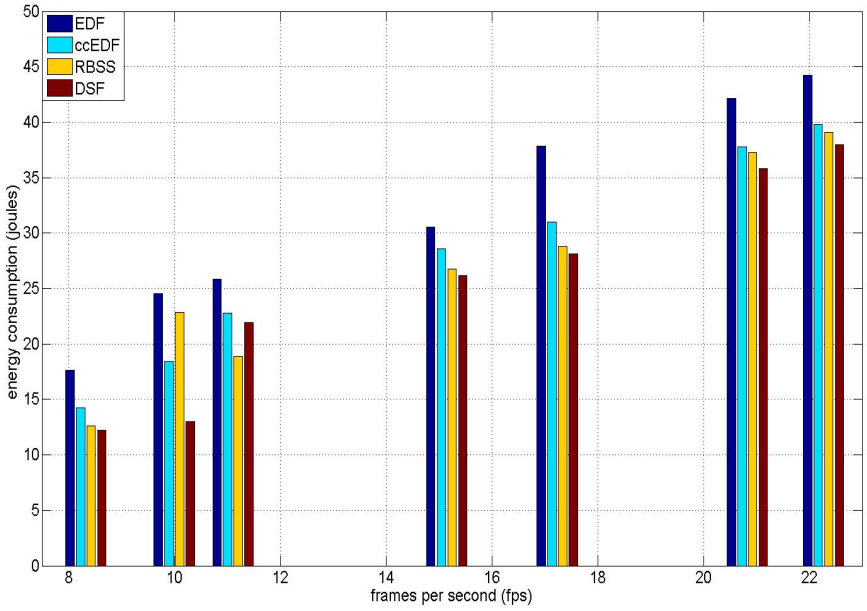


Figure 5.10: Comparative analysis of simulation results of H.264 slices version.

Figure 5.10 illustrates energy consumed under different techniques for H.264 slices version. In figure 5.10, it can be noticed that, for different frame rates, generally the DSF technique performs better than its counterparts except at 11.11fps frame rate where, RBSS technique performs better. This difference might arise due to the fact that RBSS works under the ELLF scheduling algorithm, while DSF works under the EDF algorithm. The EDF and the ELLF algorithms have different scheduling events, which can eventually result in completely different schedule of tasks. In best case, the difference in energy savings under DSF is measured up to 47% as compared to EDF, up to 29% as compared to ccEDF, and upto 43% as compared to RBSS technique. Note that only those cases are considered where there is no deadline miss under any technique. Similarly, figure 5.11 illustrates energy consumed under different techniques for H.264 pipeline version. DSF performs generally better except at 20fps frame rate where, RBSS technique performs better. This is a similar situation as for slice version. For pipeline version, best-case energy savings are measured up to 41% as compared to EDF, up to 31% as compared to ccEDF, and up to 23% as compared to RBSS.

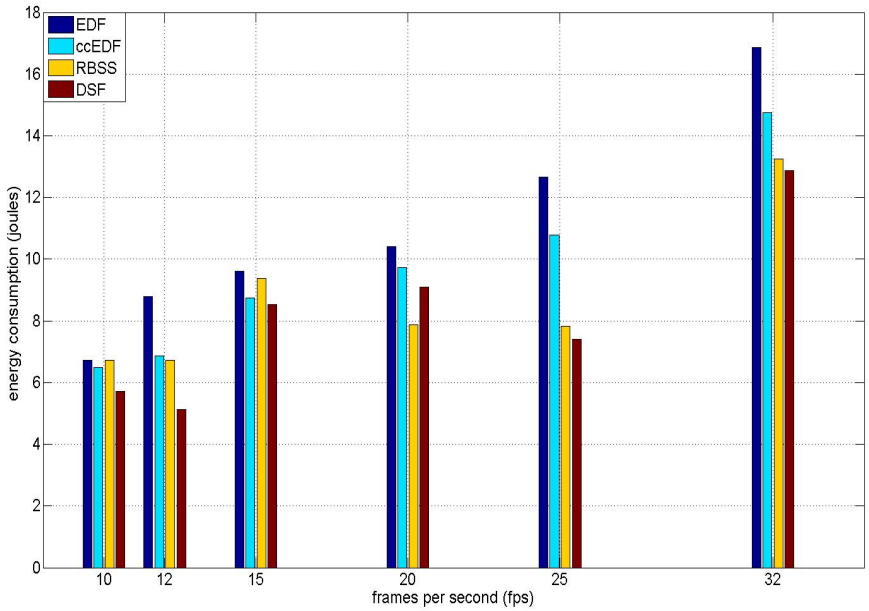


Figure 5.11: Comparative analysis of simulation results of H.264 pipeline version.

5.5 Concluding Remarks

In this chapter, we have proposed a dynamic voltage and frequency scaling technique for real-time systems, called the Deterministic Stretch-to-Fit (DSF) technique, which falls in the category of inter-task DVFS techniques. DSF comprises of an online Dynamic Slack Reclamation algorithm (DSR), an Online Speculative speed adjustment Mechanism (OSM), and an m-Task Extension technique (m-TE). The DSF technique is mainly intended for multiprocessor systems. Though, applying it on single-processor systems is also possible and in fact, rather trivial. DSF works on the principle of following the *canonical* execution of tasks, which implies that the implicit deadline guarantees available under worst-case schedulability analysis of tasks hold. That is, the aggregate utilization of tasks does not change at runtime because variations in actual execution time of tasks is compensated by proportionate variation in applied frequency by DSF. We have demonstrated in this chapter that if dynamic slack is reclaimed in such a way that no task finishes its execution later than the completion time in canonical schedule then the real-time deadlines can be guaranteed with less energy consumption. To evaluate DSF, H.264 video decoder application is taken as main use case application. Two different task models of H.264 video decoder application, which are developed at Thales Group, France [115], are used. Simulation results illustrate that the DSR algorithm, along with the OSM algorithm and the m-TE algorithm, can obtain energy savings up to 43% in best-case. Note that reported results are based on the assumption that it is possible to vary the supply voltage and operating frequency on every processor independently and over a continuous spectrum between defined lower and upper bounds. In the existing processors, however, these parameters can only be changed

in predefined discrete steps. This assumption can be lifted in this case such that if the desired optimal speed (corresponding to the achievable scalability of voltage/frequency) is not available on a processor, it has to be approximated to the closest discrete level higher than the optimal speed to prevent any deadline miss. Doing so could slightly marginalized the gains on energy. Through experiments, we illustrate that DSF is a competitive technique for dynamic voltage and frequency scaling on a multiprocessor platform. Its runtime complexity is reduced, thanks to the online construction of canonical schedule, compared to other techniques presented in [10] and [88]. The DSF technique is also integrated in a hybrid power and energy management scheme which is presented in chapter 6.

Currently, we are implementing DSF on a virtual platform for processor emulation, called QEMU [89], which is a generic and open source machine emulator and virtualizer.

Chapter 6

Hybrid Power Management Scheme for Multiprocessor Systems

Contents

6.1 Introduction

As discussed in chapter 4 and 5, dynamic power management (DPM) and dynamic voltage and frequency scaling (DVFS) policies are the most commonly used scheduling-based policies for power and energy consumption management in modern embedded systems. In this chapter, we analyze some limitations of both these policies under different operating conditions and discuss a mixed solution.

Dynamic Power management policies, as discussed in chapter 4, put system components into power-efficient states whenever they are idle due to unavailability of workload. Once applied, DPM policies eliminate both dynamic as well as static power dissipation. However, the inconvenience is that once in a power-efficient state, bringing a system component back to the active/running state requires additional energy and/or latency to service an incoming task. Dynamic voltage and frequency scaling policies, on the other hand, exploit the variations in actual workload of real-time tasks for dynamically adjusting voltage and frequency to eventually reduce dynamic power dissipation. DVFS policies are known for reducing dynamic power dissipation quite aggressively, however, there are certain inconveniences of DVFS policies due to physical limitations such as they cause increased leakage energy drain by lengthening the interval over which a computation is carried out. Due to the increasingly reduced feature-size, leakage power is becoming a significant contributing

factor in overall energy consumption. Moreover, the discrete number of voltage and frequency levels available in existing processor technology is a bottle-neck. Finding optimal real-time voltage and frequency scaling on multiprocessors is NP-hard partition problem since selectable processor frequency is discontinuous [46].

Both DPM and DVFS policies have their advantages and drawbacks. They both perform well for specific set of operating conditions such as, for particular target applications, specific architecture configuration, and/or specific scheduling algorithms. However, both policies often outperform each other when these operating conditions change [37]. For instance, a DVFS policy might perform well for applications that have large number of tasks with non-deterministic execution time of tasks (due to conditional branches) as compared to applications that have small number of tasks or more deterministic execution time of tasks. Similarly, a DPM policy might perform well on processors, which have small recovery time as compared to those who have large recovery time from power-efficient states for the same application. Thus, no single policy, whether DPM or DVFS, fits perfectly in all or most operating conditions. This leads a designer to choose for an appropriate policy every time there is a (desired or undesired) change in target application, architecture configuration, or scheduling algorithm. In this chapter, we have addressed the need of a common solution which is adaptive to changing operating conditions. Based on the fact that most of the power and energy management policies are designed for specific conditions, we propose a novel and generic scheme for energy and power management, called *Hybrid Power Management (HyPowMan)* scheme. Instead of designing new power and energy management policies to target specific operating conditions, this scheme takes a set of well-known existing policies (DPM and/or DVFS), each of which performs well for a given set of conditions, and proposes a machine-learning mechanism to adapt at runtime to the best-performing policy for any given workload. The decision of applying suitable policy is taken online and adaptive to the varying workload. The HyPowMan scheme is generic in the sense that it permits to integrate existing as well as new power and energy management policies and it can be applied under the control of global as well as partitioning-based scheduling algorithms. HyPowMan is intended mainly for multiprocessor real-time systems. However, applying it on single-processor systems is also possible. HyPowMan enhances the ability of portable embedded systems to work with larger set of operating conditions by adapting to the changing workloads and gives an overall performance that is better than any single policy can offer.

6.2 Related Work

There exists an abundant literature about the DPM and the DVFS techniques that we have previously discussed in chapter 4 and chapter 5. In this chapter, we review the state-of-the-art on some recent research work related to the interplay of DPM and DVFS policies. The growing importance of system-wide energy management clearly mandates the integration of both DPM and DVFS policies. Yet, there are not many solutions available today which use both DVFS and DPM policies and capture their intriguing trade-off in a precise way.

Authors in [126] show that both DVFS and DPM solutions often work against each other. That is, if the processing frequency is lowered through DVFS to save energy as illustrated in figure 6.1, the task execution time is extended (here, C_{dvfs} refers to elongated WCET of task due to slowdown). As a result, the idle time is shortened, which prevents DPM from putting the devices into the power-efficient states (as the idle interval may become smaller than break-even time (BET)). On the other hand, the device energy can be reduced by executing the task at higher frequency to obtain enough idle time for

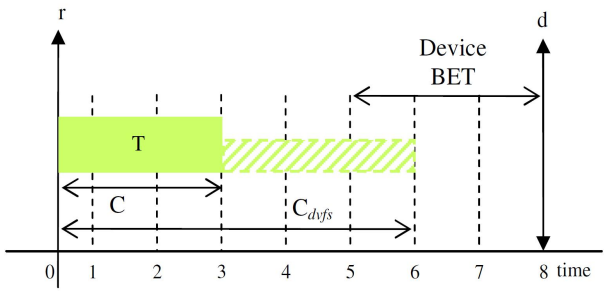


Figure 6.1: Interplay of DPM and DVFS policies.

putting the devices to the power-efficient state. However, this results in additional transition energy overhead and processor energy consumption. Moreover, the application may be using multiple devices with different power and break-even time characteristics, making an optimal solution non-trivial. While the research efforts that focus only on DVFS or DPM policies are many (as discussed in chapter 4 and 5), solutions that propose integrating both policies under a unified framework are relatively few, e.g., [27, 35, 64, 101, 131]. Authors in [101] apply a stochastic DPM policy by using the different DVFS voltage levels as multiple active power modes. The work in [131] proposes a DVFS-DPM policy that maximizes the operational lifetime of an embedded system powered by a fuel cell based hybrid power source. The frequency scaling level is chosen in [64] by investigating the trade-offs between the DVS-enabled CPU and the DPM-enabled devices. Authors in [37] propose to apply multiple DPM policies on a single-processor system in order to achieve best performance and adaptability to the varying workload. The SYS-EDF algorithm is a heuristic-based energy management scheme for periodic real-time tasks proposed in [27]. SYS-EDF applies DVFS only on processors, whereas DPM on I/O devices to save overall energy consumption. In [58], the authors introduce the concept of *critical* or *energy-efficient* speed. The energy-efficient speed is calculated by considering both the device's energy and processor's energy consumed during task executions. This stems from the observation that lowering the processor speed below a certain threshold can have negative effects on the system-wide energy consumption. Each task can potentially have a unique energy-efficient speed, depending on the devices it uses during its execution. Thus authors have proposed a single policy to manage both processor's leakage energy and device's energy in [58]. None of the previous research work has attempted to apply an entirely online and adaptive interplay of multiple DVFS and DPM policies together on an identical multiprocessor platform to manage system-wide energy while handling variable workload.

6.3 Hybrid Power Management Scheme

HyPowMan devises a top-level *policy selection mechanism*, which could select best-performing policy among the available ones for a given type of workload. This policy selection mechanism is implemented through a machine-learning algorithm. Machine-learning algorithm provides theoretical guarantee on overall performance converging to that of the best-performing policy among the available ones. This is somewhat similar to that of hybrid branch predictors employed in microprocessors and used in [37] as well. Each participating power and energy management policy is referred as an *expert* and the set of all participating

policies collectively as *expert set*. Any multiprocessor DPM or DVFS policy that can guarantee timing constraints for real-time systems is eligible to become member of expert set. When a processor is busy in executing tasks (also referred as *active* time), all DPM-based experts are inactive and are said to be *dormant* experts. However, DVFS-based experts can (or cannot) be in inactive state depending on the workload. Conversely, whenever the processor is idle, all DVFS-based experts are dormant. Any expert, which is currently active, is said to be *working* expert. A working expert solely governs all decisions related to power management on processors under the control of applied scheduling policy. Figure 6.2 illustrates how different experts are arranged by the HyPowMan scheme in an SMP architecture configuration under the control of global scheduling algorithm. For instance, expert3 in figure 6.2 is selected as working expert. Working expert returns to dormant state, which is the default state for all experts, once it finishes its job or another working expert replaces it. The most critical task for HyPowMan is to select an appropriate expert for a given power management opportunity. Power management opportunities such as idle time intervals and dynamic slack time are also referred as *input* in the remaining of this chapter. Before the machine-learning algorithm is presented in detail, we emphasize a fundamental difference between DPM-based and DVFS-based experts. The power management opportunities or input for DPM-based experts are the idle time intervals, which are inherently present in an application's schedule. Whereas, input for DVFS-based experts is dynamic slack, which is generated at runtime due to the variations in actual workload (to which, DPM-based experts can also exploit while chosen as working expert). Thus, challenges for the HyPowMan scheme are: how to measure the performance (at runtime) for different experts that work for different kind of inputs, how to evaluate them, and how to employ them in a multiprocessor context. Note that the objective of HyPowMan is to converge towards the best-performing policy *within given expert set* only and not to find *the best possible* energy savings under given operating conditions.

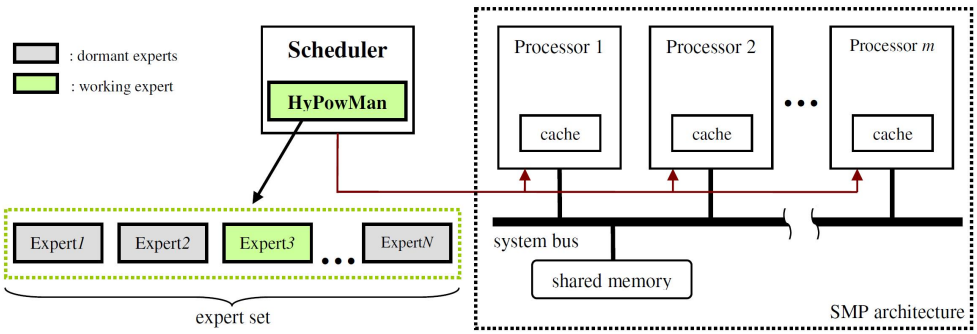


Figure 6.2: Arrangement of expert set under the HyPowMan scheme for an SMP architecture.

6.3.1 Machine-learning Algorithm

Machine-learning algorithm employed in HyPowMan, which is an adaptation of Freund and Schapire's online allocation algorithm presented in [43] and also used in [37], considers that at most N ($\forall N,\ N \geq 1$) experts are present in expert set. The algorithm associates a weight vector W^{input} with expert set, where $W^{input} = (w_1^{input}, w_2^{input}, ..., w_N^{input})$ consists of weight factors corresponding to each expert k ($\forall k, 1 \leq k \leq N$) for a given input. Every

time an input arrives, this weight vector is updated. Initially, weight factors of all individual experts are equal and sum to one in order to provide equal opportunity for all experts to perform well. However, these weights may not sum to one later during execution. HyPowMan maintains a probability vector H^{input} associated with expert set, where H^{input}= $(h_1^{input}, h_2^{input}, ..., h_N^{input})$ consists of probability factors corresponding to each expert k such that, $(0 \leq h_k^{input} \leq 1)$. This probability reflects the performance of an expert based on its weight factor. It is obtained by normalizing weight factors as shown in equation 6.1. The probability factor provides a measure on each expert's performance on previous input such that, at any point in time, the best-performing expert has the highest probability.

$$H^{input} = W^{input} / \sum_{k=1}^{N} w_k^{input} \tag{6.1}$$

HyPowMan selects expert with highest probability amongst all experts to become a working expert on next input. In case the probability factors of multiple experts are equal, working expert is chosen randomly. Once selected, a working expert governs all decisions related to power management under the control of scheduling policy. When an input is finished, the performance of all experts is evaluated. Working expert is evaluated based on how much energy was saved and how much performance degradation was incurred under that particular expert. Dormant experts are evaluated based on how they would have performed if they had been selected as working expert. This evaluation is based on the *loss* factor of each expert. Loss factor is evaluated with respect to an *ideal* (offline) power management policy that offers maximum possible energy savings and zero performance degradation. The loss factor incurred by an expert k is referred as l_k^{input}. The value of loss factor is a composition of loss in energy saving and performance degradation and it is influenced by their relative importance, which is expressed by factor α ($\forall\alpha$, $0 \leq \alpha \leq 1$). We refer to the loss factors corresponding to energy and performance as l_{ke}^{input} and l_{kp}^{input}, respectively, for expert k. Equation 6.2 represents joint loss factor of individual experts.

$$l_k^{input} = \alpha l_{ke}^{input} + (1 - \alpha) l_{kp}^{input} \tag{6.2}$$

Computation of loss factor slightly differs for DPM and DVFS experts. We elaborate this difference in section 6.3.1.1. Once the joint loss factor l_k^{input} is calculated, final step in algorithm is to update weight factors for each expert based on the loss they have incurred as shown by equation 6.3.

$$w_k^{input+1} = w_k^{input} \beta^{l_k^{input}} \tag{6.3}$$

Here, β is a constant such that $(0 \leq \beta \leq 1)$. The value of β should be set between 0 and 1 depending on the granularity of weight factors, i.e. the higher the value of β is set, the lower the variation in weight occurs for a given input. Equation 6.3 depicts that the weight factors corresponding to experts with higher loss factors are reduced while for the experts with lower loss factors are increased by simple multiplicative rule. This gives higher probability of selecting better performing experts for the next input. Note that weight and probability factors for all experts are updated once the input is terminated. Calculations related to selecting the working expert (for next input) are performed during the *active* time (i.e., the time when processors are executing tasks) and hence no additional overhead or latency is incurred when the inputs actually occur. In other words, the time consumed in updating weight and probability factors should be masked with the execution time of running tasks (for instance, in cases where dedicated hardware is used for schedulers) and therefore, when a scheduling event arrives, a working expert is already selected. HyPowMan

has linear time-complexity of the order *O(N)*, where N refers to the size of expert set. At any scheduling event, however, the overall time-complexity of the HyPowMan scheme would be bounded by the expert having largest time-complexity within expert set. This is because, after selecting an expert as working expert, HyPowMan does not participate in decision-making process. Thus, at any scheduling event, the time complexity is that of applied working expert only. During active time, when weight and probability factors are updated, the computation time is equal to the sum of all experts' computation time. This time is assumed to be masked with the execution time of running tasks. However, the computation operations still consume energy.

6.3.1.1 Loss factor for DPM and DVFS experts

For DPM experts, the input is an idle time interval. Energy loss factor (l_{ke}^{input}) is calculated by comparing the length of idle period with the time a processor has or would have spent in power-efficient state. If this time is less than the break-even time (see section 2.2.2) of processor, then there is no savings on energy and loss is maximum (l_{ke}^{input}=1). Otherwise, for sleep time greater than break-even time, equation 6.4 is used to compute l_{ke}^{input}.

$$l_{ke}^{input} = 1 - (T_{sleep-k}/T_{idle}) \tag{6.4}$$

Here, $T_{sleep-k} < T_{idle}$ and T_{idle} and $T_{sleep-k}$ refer to the length of available idle time interval and the time a processor passed in power-efficient state, respectively. Performance loss is computed based on whether a processor switched to a power-efficient state or not. If processor was transitioned to power-efficient state, the loss on performance is incurred ($l_{kp}^{input} = 1$). Otherwise, loss of performance is zero ($l_{kp}^{input} = 0$).

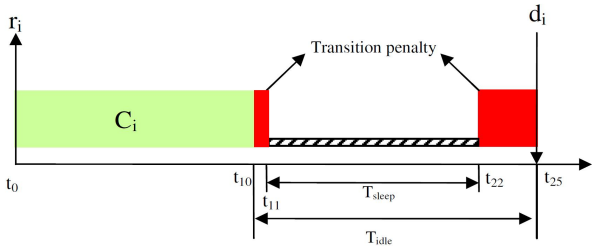


Figure 6.3: Example of the weight and probability update of a DPM-based expert.

Let us consider a simple example in figure 6.3 in which, a DPM expert is applied. Let us consider that there are at most two experts in expert set and initial weight of both experts is equal to 0.5, $\alpha = 0.60$, and $\beta = 0.80$. In the first step, length of idle period is compared with the break-even time of processor. Let us consider that the idle time (input) is sufficiently large to transition the processor in energy-efficient state. Then, in the second step, energy loss factor is computed using equation 6.4, i.e., ($l_{ke}^{input} = 1 - (11/15) = 0.266$). Since processor is transitioned, therefore, performance loss (l_{kp}^{input}) has incurred and it is equal to 1 ($l_{kp}^{input} = 1$). Based on energy and performance loss factors, weight factor of DPM expert is updated -i.e., $w_k^{input+1} = 0.88 \times w_k^{input} = 0.441$. In the last step, this updated weight factor is normalized with current weights of all other experts to eventually obtain the probability factor using equation 6.1 ($h_k^{input+1} = 0.468$). This $h_k^{input+1}$ serves HyPowMan to judge the performance of DPM expert in comparison to other expert.

For DVFS experts, the input is dynamic slack time. Similar to DPM experts, whenever there is an input available, the time it takes to scale F_{op} and V_{dd} is compared to available slack time. If the amount of slack justifies the scaling of F_{op} and V_{op} to lower values and back to F_{ref} and V_{dd} then the loss on energy is calculated using frequency scaling ratio. Otherwise, no gain on energy can be achieved by applying DVFS and thus, the loss is maximum ($l_{ke}^{input} = 1$). Performance loss is computed based on whether F_{op} and V_{op} were scaled or not. If scaling is applied, performance loss is incurred ($l_{kp}^{input} = 1$). Otherwise, no loss would have incurred ($l_{kp}^{input} = 0$).

Algorithm-9 presents our machine-learning algorithm. Suitable values for parameters (α and β corresponding to DPM and DVFS experts) should be specified by the user in accordance with the relative importance of energy and performance loss factors for soft real-time applications (lines 1-4). Initial weights, as described earlier, are equal and sum to one in the beginning. Then the expert, which offers highest probability is selected to become working expert at the start of next input (line 6). If all probability factors are equal, (which is the case at start up) then any expert is chosen randomly to become working expert. Upon the arrival of an input, the already selected working expert is applied (lines 7-8) and at the end of an input, each expert is evaluated and its weight and probability factors are updated (lines 9-11).

Algorithm 9 Machine-learning

1: **set parameters**
2: $\left\{\alpha^{dpm}, \alpha^{dvfs}\right\} \in [0,1]$
3: $\left\{\beta^{dpm}, \beta^{dvfs}\right\} \in [0,1]$
4: $w_k^1 \in [0,1] \quad (\forall k, 1 \leq k \leq N)$
5: **for all** future inputs **do**
6: select expert with maximum probability: $h^{input+1} = max\left[h_k^{input}\right]$
7: **if** input arrives **then**
8: apply selected expert as working expert
9: **else if** input terminates **then**
10: update weight vector W^{input} (apply equation 6.3)
11: update probability vector H^{input} (apply equation 6.1)

Note that HyPowMan has no direct control on the decision-making process of either experts or scheduling algorithm itself. This scheme rather functions as a top-level entity, which evaluates the performance of different experts and based on previous performance, selects best-performing expert. Once selected, its the responsibility of an expert to provide temporal guarantees for real-time tasks as long as it is a working expert. Only those policies providing real-time guarantees are chosen to be member of expert set. Whenever a DVFS-based expert is switched by any other expert, all parameters of running/preempted tasks are reinstated with respect to nominal operating conditions (V_{op}, F_{op}). Similarly, when a DPM-based expert is switched by any other expert, all processors from power-efficient state (if any) are recovered until the newly selected working expert makes any decision about their future state. Experts are switched, if necessary, only at scheduling events. Note that once an expert is selected to be a working expert by HyPowMan, it has to wait for suitable power management opportunities to appear in the application's schedule before making any decision according to its own rules. For instance, selection of a DPM expert as working expert does not imply that there are processors immediately put in power-

efficient state. Similarly, selection of a DVFS expert as working expert does not imply that processors' speed is currently scaled. To take into account the latency in recovering processors from energy-efficient states, processors are recovered before the newly selected expert starts decision-making.

6.3.2 Selection of Experts

The idea of HyPowMan is to propose an adaptive power and energy management scheme for relatively larger set of operating conditions. With the changes in workload characteristics and/or architecture configuration, it is possible that the designer/user is uncertain of the online behavior (strictly from energy consumption point of view) of a feasible target application. In such case, it becomes tricky to choose an appropriate power-management policy for changed operating conditions. In this section, we present some possible scenarios and discuss how HyPowMan can be used in each scenario.

Statically selected single expert: HyPowMan is able to apply a statically selected single expert throughout runtime. This scenario is equivalent to applying any expert designed for specific operating conditions. However, offline decision of choosing specific expert is entirely dependent on user's knowledge of application's runtime behavior and target architecture. It is trivial to select an expert if designer has sufficient knowledge of operating conditions and expert's power- and energy-efficiency. There shall be no need to adapt online to other experts.

Statically selected profiling-based experts: In this scenario, when designer does not have sufficient knowledge of target application's runtime behavior in order to determine a unique expert statically, profiling can be used to determine which experts would be suitable and in which order. In this scenario, the designer may statically apply each available expert and perform multiple iterations of simulation to determine how each expert performs during various segments of simulation time. Based on these profiles, designer can determine which experts should be used and can also specify a priori order of selected experts for different simulation segments -i.e., predefined instants at which, experts must be switched. Since statically selected single expert might not perform better throughout runtime of a target application, therefore, profiling-based selection of experts permits a target application to exploit maximum energy gains under different experts as the time evolves.

Online selected experts: This is the scenario in which profiling does not help in determining suitable expert(s) due to varying runtime behavior of target application. Machine-learning algorithm of HyPowMan is applied in this case to determine best-performing expert for different simulation time segments online and experts are switched at runtime based on how they perform. No predefined order of switching experts exists in this scenario.

6.4 Experiments

6.4.1 Setup

System model used for experiments in this chapter is the same as presented in chapter 2 and used in chapters 4 and 5. EDF scheduling algorithm is used to schedule real-time periodic tasks. Energy consumption in processors is measured under the control of each selected policy alone as well as under the control of HyPowMan. All results on energy

consumption are scaled between 0.0 and 1.0 w.r.t. the worst-case energy consumption under non-optimized algorithm within each simulation setting. That is, the energy consumed in a schedule of tasks under non-optimized algorithm, when all tasks execution with their worst-case execution time, becomes reference (1.0) and all other values, including that of non-optimized algorithm, are scaled. Moreover, it is demonstrated how the variations in α and β alter the convergence of HyPowMan towards best-performing expert. All simulations are performed for online selected experts scenario as mentioned in section 6.3.2.

6.4.2 Description of Experts

In this section, different experts that are used for experiments are described. There are three power/energy management policies (one DPM and two DVFS policies) being selected for experiments. These policies are: Timeout DPM policy, DSF (DVFS) policy, and ccEDF (DVFS) policy. In the following, we provide a brief description of these policies.

Timeout DPM policy: Timeout DPM policy is presented in [61]. In this policy, whenever there is no task to service, processor waits for a specified amount of time before transitioning to energy-efficient state. Value of timeout can be fixed or it may be changed over time statically. Timeout DPM policy is one of the most widely used policies in many applications because of its simplicity.

DSF (DVFS) policy: Deterministic Stretch-to-Fit (DSF) DVFS policy is presented in detail in chapter 5 and also in [19]. This policy is based on an online slack reclamation algorithm, which permits the dynamic slack, produced by the precedent task, to be fully consumed by single subsequent task at the appropriate priority level. Such greedy allocation of slack allows large variations in F_{op} and V_{op}, which eventually results in larger gains on energy consumption.

ccEDF (DVFS) policy: Cycle-conserving Earliest Deadline First (ccEDF) DVFS policy is presented in [87]. In this policy, the dynamic slack time is redistributed globally to all ready tasks. ccEDF considers actual execution time of all terminated tasks until their next release and worst-case execution time of all ready and running tasks to determine concurrent resource utilization of application and calculate required scaling factor for speed. Once scaling factor is determined, frequency and voltage are scaled globally on all processors.

6.4.2.1 Target application

Synthetic task sets are considered in these experiments in order to estimate energy consumption as a function of variations in three parameters –i.e., variations in total utilization $U_{sum}(\tau)$, number of tasks (n), and best-case to worst-case execution time ratio (bcet/wcet) of each task. For each data point, a task set is randomly generated, in which, each task has a uniform probability to have small ($5-25$ms), medium ($25-75$ms), or long ($75-120$ms) periods. All task periods are uniformly distributed among these three ranges. Note that a similar period generation scheme is used in [1, 87]. Individual utilization of each task u_i is also generated randomly.

6.4.3 Simulation Results

6.4.3.1 Effect of variations in bcet/wcet ratio

Simulation settings for this scenario are presented in table 6.1. The bcet/wcet ratio is varied between 50% and 100% of wcet of tasks such that *aet* of all tasks has a uniform probability distribution function as suggested in [10]. Following are the observations on these results. For bcet/wcet $ratio = 1$, figure 6.4 depicts no change in total energy consumption due to constant dynamic and static power consumption. Since *aet* remains constant, energy consumed by processors under non-optimized case and under DVFS experts remains unchanged. DPM expert, however, saves energy by exploiting the presence of inherent idle intervals. HyPowMan, in this case, converges to DPM expert in a straightforward manner as shown in figure 6.4. As bcet/wcet ratio decreases (< 1), opportunities for both DVFS experts to save energy are created as well. Figure 6.4 shows that all experts, while working alone, save energy as compared to non-optimized case. For bcet/wcet ratio between 0.5 and 0.9, it can be observed that HyPowMan converges to the best energy savings offered by either expert. This convergence validates our earlier claims that under the HyPowMan scheme, (more or less) best possible energy savings can be achieved. Figure 6.4 shows that under HyPowMan, in some cases, processors consume slightly more energy than the one offered by best-performing expert alone. This is due to the convergence mechanism in which, initially, experts are frequently switched amongst them in an attempt to figure out the best-performing expert. We have estimated best-case energy savings up to 23.12% for DSF expert alone, up to 47.94% for timout-DPM expert alone, up to 18.03% for ccEDF expert alone, and up to 47.22% for HyPowMan by interplaying all experts.

Table 6.1: Simulation settings for variable bcet/wcet ratio

Parameters	Settings
Number of processors(m) in platform	4
Number of tasks (n) in task set	8
Utilization (u_{sum}) of task set	2.50
α for DPM expert	0.80
α for DVFS experts	0.90
β for DPM expert	0.70
β for DVFS experts	0.90
bcet/wcet ratio	$50\% - 100\%$ of wcet

6.4.3.2 Effect of variations in number of tasks

Simulation settings for this scenario are presented in table 6.2. Simulations are performed by doubling and tripling the number of tasks. Results in figure 6.5 depict that, increasing the number of tasks increases the energy savings in all cases (in best-case up to 18.05% for DSF expert alone, 10.23% for timeout-DPM expert alone, 30.38% for ccEDF expert alone, and 24.71% for HyPowMan). Based on simulation results, we make two very interesting observations. Firstly, in case when all experts are applied as stand-alone policies, DVFS experts generally save more energy than DPM expert does. This is because the main determinant of variations in energy consumption is actual workload and with increased number of tasks, there are potentially more opportunities for tasks to generate dynamic slack and therefore, more possibilities for DVFS experts to reclaim energy. Secondly, HyPowMan

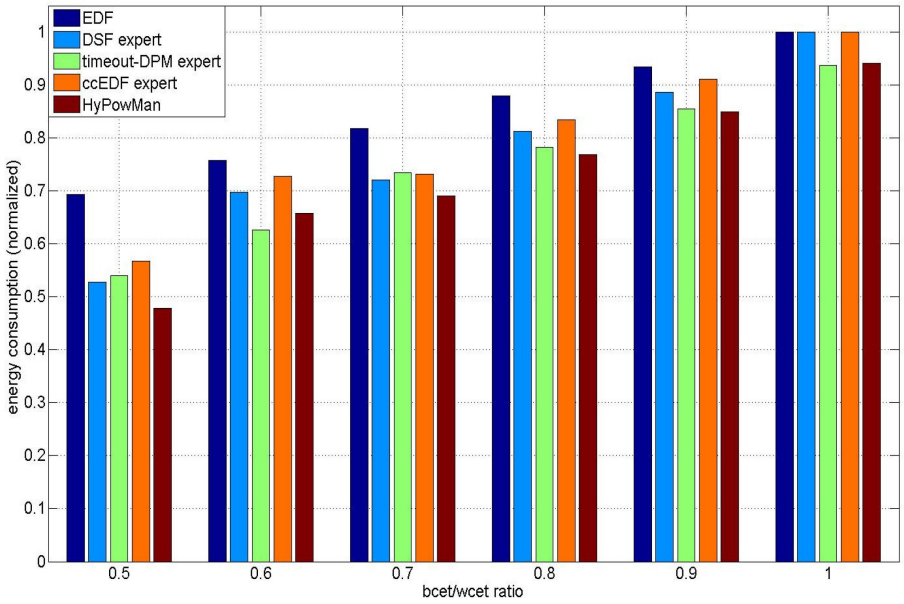


Figure 6.4: Simulation results on variation of bcet/wcet ratio.

can even result in energy savings more than any stand-alone policy in some cases. This is because, over entire simulation time, a single expert cannot always change a processor's power consumption profile (often due to transition costs). HyPowMan, on the other hand, switches an expert with the other if it is not performing well under such conditions and eventually results is better energy savings.

Table 6.2: Simulation settings for variable number of tasks

Parameters	Settings
Number of processors(m) in platform	4
Number of tasks (n) in task set	8, 16, & 24
Utilization (u_{sum}) of task set	2.50
α for DPM expert	0.80
α for DVFS experts	0.90
β for DPM expert	0.70
β for DVFS experts	0.90
bcet/wcet ratio	60% of wcet

6.4.3.3 Effect of variations in aggregate utilization

Simulation settings for this scenario are presented in table 6.3. Multiple task sets with total utilization varying between 50% (lower workload) and 100% (maximum workload) of platform capacity have been generated. Results in figure 6.6 depict that the difference in energy savings for a given utilization is more or less the same in all cases. That is, the variations in aggregate utilization do not significantly vary the performance of these policies and lesser workload naturally favors more energy savings on fixed capacity platforms.

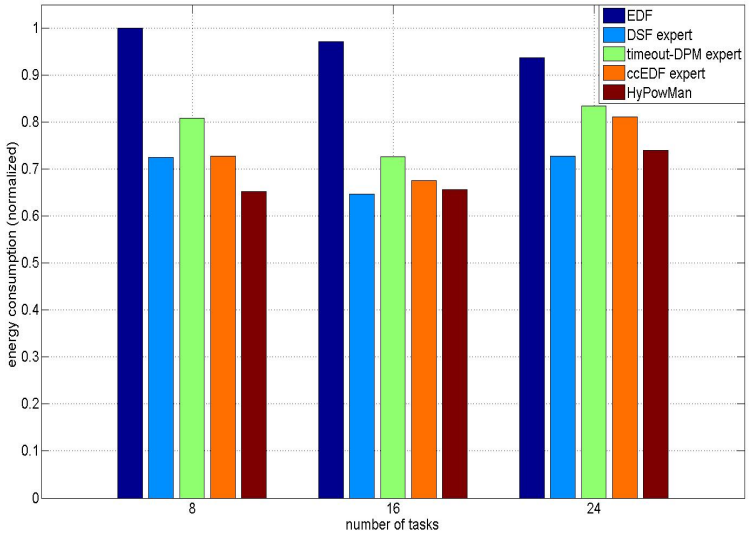


Figure 6.5: Simulation results on variation in number of tasks.

Simulation results indicate best-case energy savings compared to non-optimized case by up to 22.8% for DSF expert alone, 41.6% for timout-DPM expert alone, 34.34% for ccEDF expert alone, and 48.2% for HyPowMan for varying utilization values. Simulation results show that no particular usage pattern exists for different experts under HyPowMan and the only factor that affects the percetange of using an expert is how it performs at runtime.

Table 6.3: Simulation settings for variable aggregate utilization

Parameters	Settings
Number of processors(m) in platform	4
Number of tasks (n) in task set	8
Utilization (u_{sum}) of task set	$60\% - 100\%$
α for DPM expert	0.80
α for DVFS experts	0.90
β for DPM expert	0.70
β for DVFS experts	0.90
bcet/wcet ratio	60% of wcet

6.4.3.4 Effect of variations in α (low, medium, high)

Simulation settings for this scenario are presented in table 6.4. Recall from section 6.3.1 that value of α indicates the desirable settings of the importance of energy savings compared to the performance degradation. A high value of α indicates a higher preference to energy savings, a low value indicates higher preference to performance while a medium value indicates a reasonable ratio of both. In our experiments, we vary the value of α ranging from 0.6 (low) to 0.9 (high). We use values of α around 0.7 and 0.75 for the medium values. Results

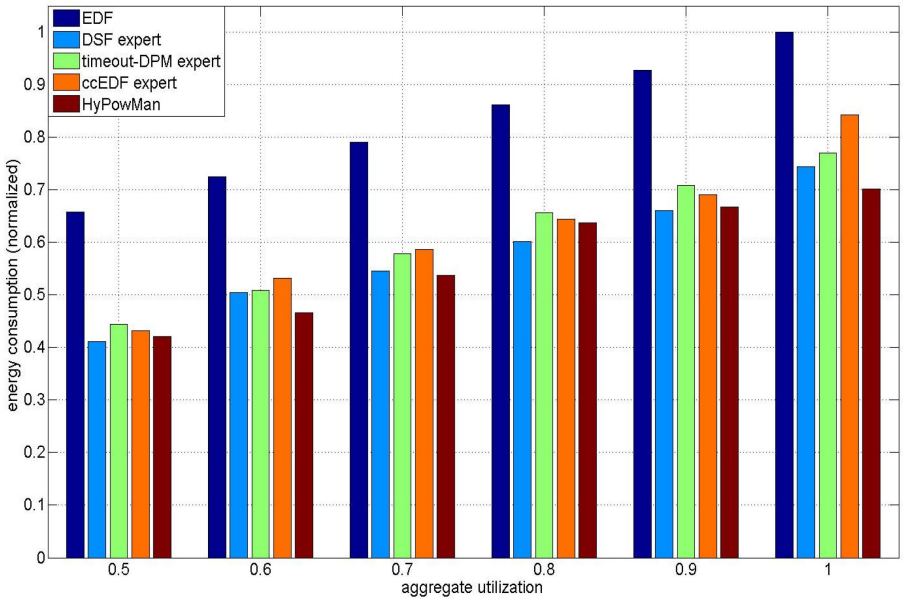


Figure 6.6: Simulation results on variation in aggregate utilization.

in figure 6.7 show that, as the value of α increases, the convergence of HyPowMan is refined with respect to energy savings -i.e., the gains achieved on energy become closer to that of best-performing individual expert. For lower values of α, the relative importance of energy savings is reduced and as a result, HyPowMan converges to the individual expert offering lesser performance degradation (an expert gaining less on energy does not transition often and therefore, performance loss is less for such experts). Note that we limit the value of α between 0.6 and 0.9. For $\alpha = 1$, performance loss will be completely overlooked, which is not realistic.

Table 6.4: Simulation settings for variable α

Parameters	Settings
Number of processors(m) in platform	4
Number of tasks (n) in task set	8
Utilization (u_{sum}) of task set	2.50
α	$0.60 - 0.90$
β for DPM expert	0.70
β for DVFS experts	0.90
bcet/wcet ratio	60% of wcet

6.4.3.5 Effect of variations in β (low, medium, high)

Simulation settings for this scenario are presented in table 6.5. Recall from section 6.3.1 that the value of β determines the granularity of weight factors, i.e. the higher the value of β is set, the lower the variation in weight of each expert occurs for a given input. From results in figure 6.8, we observe that as the value of β increases, the granularity of weight update (and hence the probability factor) associated with individual experts with respect

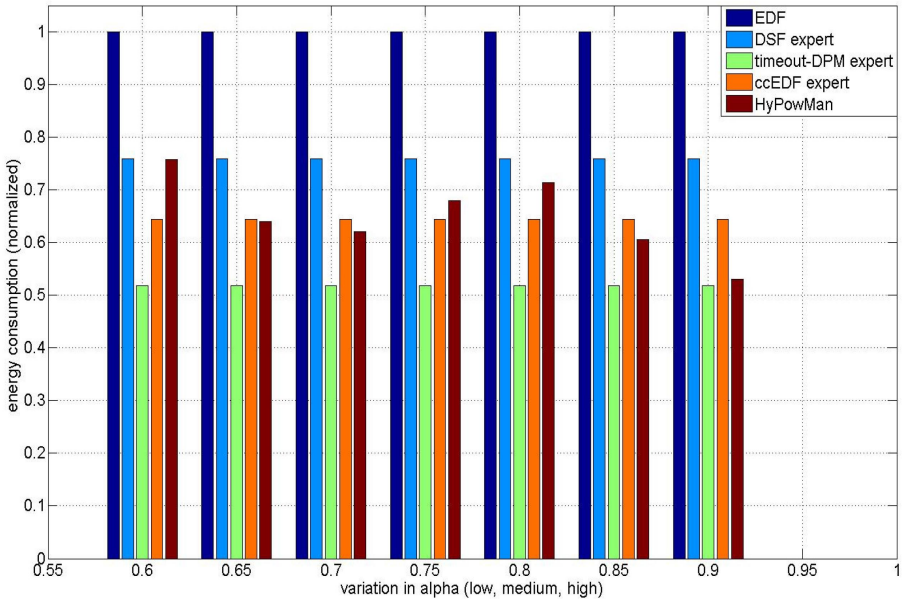


Figure 6.7: Simulation results on variation in α.

Table 6.5: Simulation settings for variable β

Parameters	Settings
Number of processors(m) in platform	4
Number of tasks (n) in task set	8
Utilization (u_{sum}) of task set	2.50
α for DPM expert	0.80
α for DVFS experts	0.90
β	$0.60 - 0.90$
bcet/wcet ratio	60% of wcet

to their performance is refined. This refinement in weight update permits HyPowMan to more precisely evaluate the performance of working expert as well as dormant experts, which eventually leads to a better convergence to best-performing expert. However, increasing β too much can result in increasing the sensitivity of HyPowMan towards weight updates and can cause increased switching of experts. Moreover, when the value of β is decreased too much, HyPowMan has lesser precision in weight updates and therefore, frequently switches the working expert again, which eventually leads to lesser gains on energy and more performance loss.

6.5 Concluding Remarks

In this chapter, we have proposed a generic power and energy management scheme for multiprocessor systems, called Hybrid Power Management (HyPowMan) scheme. HyPowMan serves as a top-level entity which, instead of designing new power management policies (whether DPM or DVFS) for specific operating conditions, takes a set of well-known existing policies, each of which performs well for a given set of conditions, and adapts at runtime to

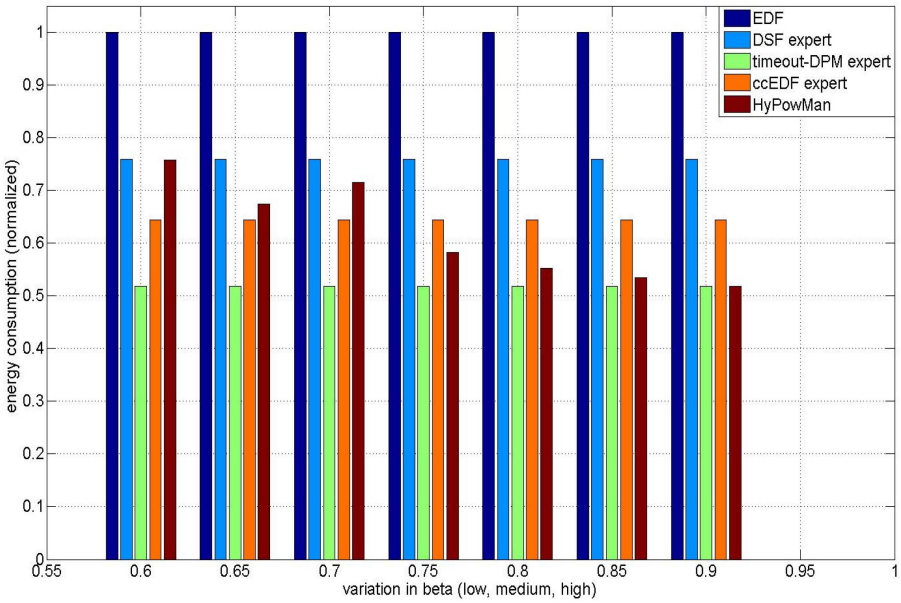


Figure 6.8: Simulation results on variation in β.

the best-performing policy for any given workload. A machine-learning algorithm is implemented to evaluate the performance of all policies and the decision to select best-performing policy at any point in time is taken online and adaptive to the varying workload. Experiments validate that our proposed scheme adapts well to the changing workload and quickly converges to the best-performing policy within the selected expert set by a margin of 1.5% to 11% (w.r.t. best-case) on energy savings. Through experiments, we have validated that HyPowMan is robust to variations in application task parameters such as actual execution time, number of tasks, and aggregate utilization. The expert set used for experiments in this chapter is composed of both DPM and DVFS policies. However, we have also validated the working of HyPowMan by using similar policies only –i.e., only DVFS or only DPM policies. Moreover, experiments using AsDPM policy (chapter 4), along with DSF policy (chapter 5), were also performed. Interested reader can find those results in Appendix B. One of the limitations of HyPowMan is that it evaluates the performance of all experts (working as well as dormant) for every input (i.e. power- and energy-saving opportunity), which may increase the computational overhead if larger expert sets are used. This aspect requires improvements for an efficient implementation on hardware platforms.

Chapter 7

Conclusions and Future Research Perspectives

Real-time embedded systems have become ubiquitous in our life, ranging from hand-held cell phones and home appliances to more sophisticated systems like data centers, signal-processing systems, satellites, and military applications. As the uses for real-time systems become diverse, the research on these systems has confronted with many emerging challenges. Nowadays, real-time applications have become sophisticated and complex in their behavior and interaction. Their sophistication and complexity has led to the emergence of multiprocessor architectures. Another reason for multiprocessor architectures to be readily acceptable in commercial sector is that they have been considered as a solution to the *thermal roadblock* imposed by single-processor architectures. It has become clearer that in future, most of the sophisticated real-time applications will be deployed on multiprocessor platforms. Inevitably, the change in platform architecture design and complexity of real-time applications have renewed some existing challenges as well as brought some new ones for real-time research community. One of the major renewed challenge is that single-processor optimal scheduling algorithms can not be applied on multiprocessor systems without loss of optimality. Thus, the real-time research community has to develop alternative scheduling strategies to incorporate the aspects of multiprocessor systems. One of the new challenges that real-time systems are facing is to reduce power and energy consumption while maintaining assurance that timing constraints will be met. As the computational demands of real-time embedded systems continue to grow, effective yet transparent energy-management approaches are becoming increasingly important to minimize energy consumption, extend battery life, and reduce thermal effects. Power and energy consumption in real-time systems has earned great importance over the last few years and many software-based approaches to statically as well as dynamically reduce power and energy consumption have been proposed. Software-based solutions, particularly energy-aware scheduling techniques such as dynamic voltage & frequency scaling and dynamic power management techniques have emerged. Furthermore, system-wide energy-efficient techniques that include other system components such as memory sub-systems and interconnect network have been proposed as well. Yet their flexibility is often matched by the complexity of the solution.

This dissertation is an attempt to ameliorate energy-management in real-time embedded systems by proposing new flexible and effective energy-aware scheduling strategies. Techniques proposed in this dissertation increase the set of operating conditions in which, real-time applications can be scheduled in an energy-efficient manner while maintaining their temporal guarantees on multiprocessor platforms. We divide this chapter into two parts. First part summarizes our contributions and the significance of results obtained. The second part presents some future research perspectives and ameliorations related to this dissertation.

7.1 Summary of Contributions and Results

In this section, we summarize the contributions of this dissertation.

Two-Level Hierarchical Scheduling Algorithm. Restricted-migration scheduling strategies provide a good compromise between the full migration and the partitioning strategies [24, 63, 62]. It is flexible enough to allow dynamic tasks to join the system at runtime, but it does not incur large migration overheads as compared to full-migration strategies. In this dissertation, in chapter 3, we have presented a multiprocessor scheduling algorithm, called two-level hierarchical scheduling algorithm (2L-HiSA), which falls in the category of restricted migration scheduling. The EDF scheduling algorithm has the least runtime complexity among job-level fixed-priority algorithms for scheduling tasks on multiprocessor architecture. However, EDF suffers from sub-optimality in multiprocessor systems. 2L-HiSA addresses the sub-optimality of EDF as global scheduling algorithm and divides the problem into a two-level hierarchy of schedulers. 2L-HiSA uses *multiple instances* of single-processor optimal EDF scheduling algorithm in a hierarchical fashion at two levels: an instance of EDF at top-level scheduler and an instance at local-level scheduler on every processor of the platform.

Moreover, 2L-HiSA consists of two phases: 1) the task-partitioning phase, in which, each non-migrating task is assigned to a specific processor by following the bin-packing approach, 2) the processor-grouping phase, in which, processors are grouped together based on their workload characteristics. 2L-HiSA partitions application tasks statically onto processors by following the bin-packing approach, as long as schedulability of tasks partitioned on a particular processor is not violated. Tasks that can not be partitioned on any processor in the platform qualify as migrating or global tasks. Furthermore, it makes clusters of identical processors such that, per cluster, the unused fragmented computation power equivalent to at most one processor is available. We show that 2L-HiSA improves on the schedulability bound of restricted-migration based scheduling using multiple instances of preemptive EDF for multiprocessor systems. 2L-HiSA can ensure deadline guarantees if a subset of tasks can be partitioned such that the under-utilization per cluster of processors remain less than or equal to the equivalent of one processor. Partitioning tasks on processors reduces scheduling related overheads such as context switch, preemptions, and migrations, which eventually help reducing overall energy consumption. The NP-hardness of partitioning problem [60], however, can often be a limiting factor. By using clusters of processors instead of considering individual processors, 2L-HiSA alleviates bin-packing limitations by effectively increasing *bin* sizes in comparison to *item* sizes. With a cluster of processors, it is much easier to obtain the unused processing power per cluster less than or equal to one processor. We provide simulation results to support our proposition. We have illustrated that power- and energy-efficient strategies like DVFS and/or DPM can be used in conjunction with 2L-HiSA to improve energy savings. Furthermore, we have illustrated that the task preemption- and migration-related overhead is significantly less when using 2L-HiSA as compared to PFair and ASEDZL, which are multiprocessor optimal scheduling algorithms.

Assertive Dynamic Power Management Technique. Another contribution of this dissertation, presented in chapter 4, is the proposition of a dynamic power management technique for multiprocessor real-time systems. DPM techniques achieve energy conservation in embedded computing systems by actively changing the power consumption profile of the system by selectively putting its components into power-efficient states sufficient to meeting functionality requirements. Our proposed technique is called *Assertive* Dynamic

Power Management (AsDPM) technique because of its aggressiveness towards the extraction of the idle time intervals, if present, from the application's runtime schedule which is not the case in conventional DPM techniques. AsDPM technique works under the control of global EDF and global LLF scheduling algorithms. It is an admission control technique for real-time tasks which makes a feasible task set energy efficient by deciding when exactly a ready task shall execute. Without this admission control, all ready tasks are executed as soon as there are enough computing resources (processors) available in the system, leading to poor possibilities of putting some processors into power-efficient states. AsDPM technique differs from the existing DPM techniques in the way it exploits the idle time intervals. A conventional DPM technique can exploit idle intervals *only* once they occur on a processor –i.e., once an idle interval is detected. Upon detecting idle time intervals, these techniques decide whether to transition target processor(s) to power-efficient state. AsDPM technique, on the other hand, aggressively extracts all idle intervals from some processors and clusters them on some other processors of the platform to elongate the duration of idle time. Transitioning processors to suitable power-efficient state then become a matter of comparing idle interval's length against the break-even time (BET) of target processor. Although, AsDPM is an online dynamic power management technique, its working principle can be used to determine static optimal architecture configurations (i.e., number of processors and their corresponding voltage-frequency level, which is required to meet real-time constraints in worst-case with minimum energy consumption) for target application through simulations. In this dissertation, we have demonstrated the use of AsDPM for determining static optimal architecture configurations at first, and then as an online DPM technique, which can further increase energy-efficiency of already (statically) optimized configurations.

Deterministic Stretch-to-Fit DVFS Technique. Dynamic voltage and frequency scaling is one of the effective techniques, which aim at changing energy consumption profile of real-time embedded systems. This is because energy consumption of processors is a quadratic function of supply voltage of processors [48]. Real-time applications potentially exhibit variations in their actual execution time and therefore, often finish much earlier than their estimated worst-case execution time. Real-time DVFS techniques exploit these variations in actual workload for dynamically adjusting the voltage and frequency of processors in order to reduce power consumption. In this dissertation, in chapter 5, we propose an *inter-task* dynamic voltage and frequency scaling technique for real-time multiprocessor systems, called *Deterministic Stretch-to-Fit (DSF)* technique. The DSF technique comprises three algorithms, namely, *Dynamic Slack Reclamation (DSR)* algorithm, *Online Speculative speed adjustment Mechanism (OSM)*, and *m-Tasks Extension (m-TE)* algorithm. The DSR algorithm is the principle slack reclamation algorithm of DSF that assigns dynamic slack, produced by a precedent task, to the appropriate priority next ready task that would execute on the same processor. While using DSR, dynamic slack is not shared with other processors in the system. Rather, slack is fully consumed on the same processor by the task, to which it is once attributed. Such greedy allocation of slack allows the DSR algorithm to have large slowdown factor for scaling voltage and frequency for a single task, which eventually results in improved energy savings. DSR works in conjunction with *global* scheduling algorithms on identical multiprocessor real-time systems. The OSM and the m-TE algorithms are extensions of the DSR algorithm. The OSM algorithm is an online, adaptive, and speculative speed adjustment mechanism, which anticipates early completion of tasks and performs *aggressive* slowdown on processor speed. Apart from saving more energy as compared to the stand-alone DSR algorithm, this speculative speed adjustment mechanism also helps to avoid radical changes in operating frequency and supply voltage,

which results in reduced peak power consumption, which leads to an increase in battery life for portable embedded systems. The m-TE algorithm extends an already existing *One-Task Extension (OTE)* technique for single-processor systems onto multiprocessor systems.

DSF works on the principle of following the *canonical* execution of tasks, which implies that the implicit deadline guarantees available under worst-case schedulability analysis of tasks hold. That is, the aggregate utilization of tasks does not change at runtime because variations in actual execution time of tasks is compensated by proportionate variation in applied frequency by DSF. We have demonstrated in this chapter that if dynamic slack is reclaimed in such a way that no task finishes its execution later than the completion time in canonical schedule then the real-time deadlines can be guaranteed with less energy consumption. To evaluate DSF, H.264 video decoder application is taken as main use case application. Two different task models of H.264 video decoder application, which are developed at Thales Group, France [115], are used. Simulation results illustrate that the DSR algorithm, along with the OSM algorithm and the m-TE algorithm, can obtain energy savings up to 43% in best-case. Note that reported results are based on the assumption that it is possible to vary the supply voltage and operating frequency on every processor independently and over a continuous spectrum between defined lower and upper bounds. In section 7.2, we discuss how removing this assumption can effect the working of DSF.

Hybrid Power Management Technique. Both dynamic power management (DPM) and dynamic voltage & frequency scaling (DVFS) policies are the most commonly used scheduling-based policies for power/energy consumption management in modern embedded systems. They both perform well for specific set of operating conditions such as, for particular target applications, specific architecture configuration, and/or specific scheduling algorithms. However, both policies often outperform each other when these operating conditions change. Thus, no single policy, whether DPM or DVFS, fits perfectly in all or most operating conditions, leading a designer to choose for an appropriate policy every time there is a change in target application, architecture configuration, or scheduling algorithm. In this dissertation, in chapter 6, we have addressed the need of a common solution which is adaptive to changing operating conditions. Based on the fact that most of the power/energy management policies are designed for specific conditions, we propose a generic power/energy management scheme called Hybrid Power Management (HyPowMan) scheme. Instead of designing new power/energy management policies (DPM or DVFS) to target specific operating conditions, HyPowMan scheme takes a set of well-known existing policies, each of which performs well for a given set of conditions, and proposes a machine-learning mechanism to adapt at runtime to the best-performing policy for any given workload. The decision of applying suitable policy is taken online and adaptive to the varying workload. HyPowMan scheme is generic in the sense that it permits to integrate existing as well as new power/energy management policies and it can be applied under the control of global as well as partitioning-based scheduling algorithms. This scheme serves as a top-level entity of all power management policies within an expert-set. It does not intervene in decision-making process of either controlling scheduling algorithm or applied (working) expert. The job of HyPowMan scheme is only to select a best-performing expert from expert-set based on its performance. Once selected, an expert makes all power management decisions under the control of scheduling algorithm. Thus, HyPowMan scheme enhances the ability of embedded real-time systems to adapt with changing workload by working with a larger set of operating conditions and gives overall performance and energy savings that are better than any single policy can offer. Utility of HyPowMan scheme can be extended beyond energy-efficiency –i.e., the adaptivity of

HyPowMan scheme allows to define other performance parameters as principle criteria for converging to the best-performing expert. For instance, peak power consumption, battery life, and temperature threshold can be the principle criteria for HyPowMan scheme for adapting to the best-performing expert.

7.2 Future Research Perspectives

Real-time systems have very complex characteristics. Changing one aspect of the system can result in a very different problem. In this dissertation, we have focused mainly on achieving the energy-efficiency in real-time multiprocessor systems through scheduling. Our contributions deal with specific models of tasks and processing platform which are discussed in chapter 2. Generalization of these models expands the number of possible systems that can be used by the real-time applications. Therefore, further extensions of this work include generalizing the task model and processing model. Many of these generalized models will require certain ameliorations in our proposed techniques that we discuss in this section.

7.2.1 Task Models

Inter-Task Dependency. The research work presented in this dissertation assumes an independent and preemptive task model –i.e., the execution of one task's job is not contingent upon the status of another task's job. There are many problems in which independence is an unreasonable assumption. Also, in some cases preemption may not be allowed. Jobs may have dependencies for a variety of reasons. Two types of job dependencies are *resource sharing* and *precedence constraints*. Incorporating resource sharing and precedence constraints into the results presented in this dissertation would be a natural extension. However, dependencies between tasks can cause priority inversions –situations where a higher priority job is blocked while a lower priority job executes. Such priority inversions may violate our assumptions. For instance, in case of 2L-HiSA scheduling algorithm, two tasks with precedence constraint may require to be statically assigned on the same processor, thus adding another limitation along with the partitioning problem. Similarly, in case of AsDPM technique, resource sharing between tasks may cause a processor to stand still in idle state that could have been otherwise, transitioned to power-efficient state as soon as the runtime workload is reduced.

Task Preemption. Task preemption is a common assumption in real-time systems, however, there are situations when jobs may not be preempted. Removing the ability to preempt jobs may cause priority inversions. Moreover, non-preemptive systems can also suffer from scheduling anomalies whereby a feasible system may miss deadlines if one or more jobs are removed from the system or complete executing early. Accounting for priority inversions and scheduling anomalies can be an important extension to the research presented in this dissertation. However, we did not consider non-preemptive systems. Moreover, we have considered only feasible task systems with no scheduling anomalies.

Period of Task. Another common assumption in real-time systems is about the task models based on their periodicity. This parameter can be interpreted in three distinct ways, each of which leads to a well-defined type of task. According to the interpretation given to the period, tasks can be classified into three categories: periodic task model, sporadic task model, and aperiodic task model. In this dissertation, we have considered periodic task

models –i.e., every task has an exact inter-arrival time between successive jobs. Extending our proposed techniques to sporadic and aperiodic task models would be an interesting contribution as it would allow the handling of a wide range of real-time applications.

7.2.2 Platform Architectures

This dissertation considers homogeneous multiprocessor platform of type SMP (Symmetric shared-memory Multiprocessor) composed of identical processors. There are a variety of ways in which the processing platform model can be generalized. For instance, uniform multiprocessor or unrelated multiprocessor platforms can be used instead of identical multiprocessor platform. Certain contributions of this dissertation, such as AsDPM and DSF techniques, have been used in French national project Pherma [86]. This project proposes a heterogeneous CMP (Chip Multi-Processing) processing platform model called SCMP (Scalable Chip Multi-Processing) that supports dynamic migration and preemption of tasks by using physically distributed, logically shared memories [120]. The heterogeneity in SCMP platform is *inter-cluster* of processing units (i.e., multiple clusters/groups of identical processors are formed, however, different clusters can contain different type of processing units). SCMP is good alternative platform compared to SMP platform for further extensions of our proposed techniques. Furthermore, the results presented in this dissertation consider mostly the power and energy consumption of processors only and did not include energy consumed by other sub-systems such as memory and interconnect network. Recent research reports that peripheral devices, memory subsystems, flash drives, and wireless network interfaces are pervasive in modern embedded systems that consume considerable amount of energy. In order to achieve system-wide energy savings, energy-aware I/O scheduling algorithms need to be developed for real-time systems. Extension of our proposed techniques to system-wide energy-efficient techniques is a possible direction. For instance, we have demonstrated in chapter 4 that AsDPM technique can be useful for memory subsystems. HyPowMan scheme can also incorporate the energy consumption of peripheral devices and other subsystems while selecting suitable experts.

Another aspect concerning the generalization of processing platform is the method of scaling voltage & frequency in processors that support multiple voltage-frequency levels. In this dissertation, we have considered that it is possible to vary the voltage & frequency on every processor independently and over a continuous spectrum between defined lower and upper bounds. This consideration is based on the fact that processors that are able to operate on a (more or less) continuous voltage and frequency spectrum are fast becoming a reality [10]. However, this assumption can be lifted for processors offering only discrete voltage-frequency levels such that if the (calculated) optimal processor speed is not available on a processor, it has to be approximated to the closest discrete level higher than the optimal speed to prevent any deadline miss. This conversion is straightforward for tasks that satisfy sufficient schedulability bound. For a feasible task set satisfying only necessary schedulability condition, converting to discrete operating frequency would mean variation in concurrent utilization at runtime, which could lead to deadline misses. This aspect would require slight ameliorations for DSF technique for instance, which is based on the principle of following canonical execution of tasks.

7.2.3 Scheduling Algorithms

Most of the contributions presented in this dissertation consider the earliest deadline first (EDF) scheduling algorithm for scheduling real-time tasks on a multiprocessor platform. Even though there are many reasons why EDF is a reasonable scheduling algorithm to

consider, it is pertinent to ask whether the contributions presented in this dissertation can work in conjunction with other scheduling algorithms as well. We have discussed in chapter 3 that EDF algorithm has the least runtime complexity for scheduling tasks on multiprocessor architectures, but it suffers from sub-optimality in multiprocessor context. To improve the schedulability bound of EDF-based systems, we have proposed 2L-HiSA algorithm which uses multiple instances of single-processor optimal EDF algorithm in a hierarchy of schedulers. For the remaining contributions of this dissertation, extending them while using other algorithms would be a valuable future contribution. We discuss the possibilities of extending our proposed techniques to other multiprocessor scheduling algorithms in the following.

AsDPM Technique. Extending AsDPM technique to any of the multiprocessor scheduling algorithm which is based on the *fluid scheduling mechanism* (for instance, PFair [13] and LLREF [28] algorithms) is not possible due to their contradiction with the working principle of AsDPM technique. In algorithms based on fluid scheduling mechanism, all ready tasks should execute proportionately to their respective utilization (or so called weight). In AsDPM technique, on the other hand, a ready task can be deferred (delayed) from execution until and unless its anticipative laxity becomes negative (see chapter 4), which is contrary to the working principles of LLREF, PFair, and its heuristic algorithms. It is possible, however, to extend AsDPM technique to other algorithms such as LLF [71].

DSF Technique. It is possible to extend all three algorithms –i.e., DSR, OSM, and m-TE, presented under DSF technique to other scheduling algorithms. Since DSF technique works on the principle of following canonical execution of tasks, therefore, it is possible to integrate DSF technique with other algorithms with slight changes in the method of calculating and attributing dynamic slack at runtime.

HyPowMan Scheme. HyPowMan scheme itself does not intervene in the decision-making process of either scheduling algorithm or selected working expert (power-management strategy). Thus, it is possible to implement HyPowMan scheme with other scheduling algorithms. The only limiting factor can be the overall time complexity of the system. For instance, applying HyPowMan scheme with PFair algorithm, which is known to be complex due to its scheduling overheads, would considerably increase system's time complexity.

7.2.4 Implementation strategy –Simulations vs Real Platforms

In this dissertation, we have relied mainly on simulations for the evaluation and validation of our proposed techniques. Our motivation for using simulation-based approach comes from the fact that, initially, it is difficult to validate the analytical results if the assumptions on task and platform models are not satisfied. Satisfying assumptions on system model using real platforms is not straightforward. Furthermore, in simulations, it is easier to approximate the scheduling- and performance-related overheads of proposed energy-management techniques by varying system configurations such as processor type, cache behavior, inter-processor communications, inter-task dependencies, workload variations, and scheduling algorithms. We have used a modern simulation tool called STORM [108] for our simulations. STORM is a free-ware Java-based simulation tool for multiprocessor scheduling algorithms incorporating both hardware and software architecture parameters (see Appendix A for details). After evaluating the performance of our proposed techniques using simulations,

we are already focusing on their implementation on a real platform. The platform we are using for development is from the ARM® ($ARM1176JZF-S$) [9]. This platform offers TrustZone® and the Intelligent Energy Management (IEM) technologies. Moreover, we are currently implementing AsDPM and DSF techniques using virtual platform for processor emulation called QEMU [89]. QEMU is a generic and open source machine emulator and virtualizer. It provides a set of device models, allowing it to run a variety of unmodified guest operating systems; it can thus be viewed as a hosted virtual machine monitor.

7.2.5 Thermal Aspects

One of the new challenges faced by the real-time systems is the increasing per-chip transistor-density due to reduced feature-sizes. The increased chip-density and high-speed computation requirements further lead to the increase in heat dissipation in multiprocessor systems as well. This increase in temperature causes the creation of *hot-spots*, which greatly reduces the component's shelf-life. Moreover, designers are now focusing on fabrication of 3D-stacked architectures for multi-core platforms [30, 104] in order to satisfy ever-increasing computation requirements. Due to scaling-down of transistor, the available chip surface for heat dissipation is reducing which results in increased power-densities as the leakage current is not scaled down with the same factor. Multiprocessor systems also behave as multiple heat sources which increase the likelihood of temperature variations over shorter time and chip area rather than just a uniform temperature distribution over entire die [78]. The energy consumed in computing devices is in large part converted into heat. Thus, thermal imbalances, along with the power and energy consumption, have become a first-class design consideration for modern embedded systems. The system model considered in this dissertation contain power and energy consumption models, which are somewhat independent of thermal aspects. This would most definitely be an interesting and useful contribution to extend our proposed techniques to incorporate thermal aspects. In chapter 4, we have briefly described the handling of thermal imbalances that can appear due to AsDPM technique. Clearly, more precise and thorough investigation of thermal aspects is required.

7.3 Summary

The emergence of multi-core technology has brought a paradigm shift for the research on real-time embedded systems. Real-time applications that run upon multiprocessor platforms are likely to be extremely diverse and characterized by complex software behavior and interactions. The sophistication and the complexity of real-time applications that run upon multiprocessor platforms have renewed, among many other challenges, the challenge of power and energy consumption optimization, while providing assurance that timing constraints will be met. This dissertation is based on the thesis that energy-efficiency and scheduling of real-time systems are closely related problems, which should be tackled together for best results. The contributions proposed in this dissertation ameliorate, through scheduling, the energy-efficiency of real-time systems that can be proven predictable and temporally correct over multiprocessor platforms. Our proposed solutions are flexible to varying system requirements, less complex, and effective. Future research will continue to remove some of the simplifying assumptions from the real-time task models and the platform architecture that we have used. Moreover, we have highlighted that system-wide energy-efficiency and thermal effects should be taken into account for best results.

Publication List

Refereed Journal Papers

1. **M. K. Bhatti**, C. Belleudy, M. Auguin, "Hybrid Power Management in Real-time Embedded Systems: An Interplay of DVFS and DPM Techniques", *In Springer's journal on Real-time Systems (RTS), special issue on Temperature/Energy Aware Real-Time Systems. DOI:* $10.1007/s11241 - 011 - 9116 - y$. *(To appear in early 2011).*
2. **M. K. Bhatti**, C. Belleudy, M. Auguin, "Two-level Hierarchical Scheduling Algorithm for Real-time Multiprocessor Systems", *In Journal of Software (JSW), Academy Publishers. (Accepted for publication).*

Refereed Conference Papers

1. **M. K. Bhatti**, M. Farooq, C. Belleudy, M. Auguin, O. Mbarek, "Assertive Dynamic Power Management (AsDPM) Strategy for Globally Scheduled RT Multiprocessor Systems", *In the proceedings of Power and Timing Modeling, Optimization and Simulation, PATMOS'*09*, and Integrated Circuit and System Design, chapter* 8*, Springer LNCS Vol.* 5953/2010*, ISBN* $978 - 3 - 642 - 11801 - 2$*, Pages* $116 - 126$*,* 2010.
2. **M. K. Bhatti**, C. Belleudy, M. Auguin, "Power Management in Real-time Embedded Systems through Online and Adaptive Interplay of DPM and DVFS Policies", *In the proceedings of International Conference on Embedded and Ubiquitous Computing, EUC'*10*, December* 2010*, Hong Kong, SAR, China.*
3. **M. K. Bhatti**, C. Belleudy, M. Auguin, "An Inter-Task Real-time DVFS Scheme for Multiprocessor Embedded Systems", *In the proceedings of International Conference on Design and Architectures for Signal and Image Processing, DASIP'*10*, October* 2010*, Edinburgh, UK.*
4. K. Ben Chehida, R. David, F. Thabet, **M. K. Bhatti**, M. Auguin, C. Belleudy, A.M. Déplanche, Y. Trinquet, R. Urunuela, , F. Broekaert, V.Seignole, A. M. Fouillart, "PHERMA, A global approach for system-level energy consumption optimization for Real-time heterogeneous MPSoC architectures", *In Proceedings of Low Voltage & Low Power Consumption Conference, FTFC'*09*,* 2009*, Neuchâtel, Switzerland.*
5. **M. K. Bhatt**i, C. Belleudy, M. Auguin, "A Framework for Offline Optimization of Energy Consumption in Real-time Multiprocessor System-on-Chip", *In the proceedings of IEEE International Conference on Electronics, Circuits, and Systems, ICECS'*09*, December* 2009*, Hammamet, Tunisia.*
6. **M. K. Bhatt**i, M. Farooq, C. Belleudy, M. Auguin, "Improving resource utilization under EDF-based mixed scheduling in multiprocessors real-time systems", *In the proceedings of IFIP/IEEE International Conference on Very Large Scale Integration, VLSI-SoC'*08*,* 2008*, Rhodes Island, Greece.*
7. M. Farooq, **M. K. Bhatti**, F. Muller, C. Belleudy, M. Auguin, " Precognitive DVFS: Minimizing Switching Points to Further Reduce the Energy Consumption", *In the proceedings of* 14*th IEEE Real-Time and Embedded Technology and Applications Symposium,* 2008*, St. Louis, MO, USA.*

Non-Refereed Workshop/Conference Papers

1. **M. K. Bhatti**, C. Belleudy, M. Auguin, "Controlling Energy Profile of Real-time Multiprocessor Systems by Anticipating Application's Workload at Runtime", *In the proceedings of SYMPosium on new Machine Architectures SympA'13, 2009, Toulouse, France.*
2. **M. K. Bhatti**, C. Belleudy, M. Auguin, "A hybrid scheduling algorithm for optimizing resource utilization & energy consumption in RT multiprocessor systems", *In the proceedings of 3rd National Symposium of Research Group on System-on-Chip, System-in-Package, GDR SoC − SiP'08, 2008, Paris, France.*
3. **M. K. Bhatti**, M. Farooq, C. Belleudy, M. Auguin, "Mixed Scheduling for Improved Resource Utilization and Energy Consumption in Real-time Multiprocessor Systems", *In the proceedings of Sophia Antipolis MicroElectronics, SAME'08, 2008, Sophia Antipolis, France.*

Part II

Selected chapters: French version

Chapter 1

Introduction

Contents

1.1 Introduction

Dans les systèmes temps réel, le respect des contraintes temporelles est aussi important que la validité des résultats [42]. Ces systèmes doivent donc non seulement effectuer des opérations correctes, mais aussi les exécuter au bon moment. Une opération logiquement correcte peut produire une sortie erronée, qui devient alors totalement inutile, ou dégradée en fonction de la sévérité des contraintes temporelles. Selon le niveau de sévérité des contraintes temporelles, les systèmes temps réel peuvent être classés en trois grandes catégories: les systèmes temps réel dur, souple ou ferme [47, 77, 105]. Ces systèmes doivent être prédictibles et il est nécessaire de prouver le respect des échéances temporelles. Le concepteur doit vérifier que le système soit correct avant son implémentation, c'est-â-dire, par exemple, pour toutes les exécutions possibles d'un système temps réel dur, les échéances temporelles doivent être respectées. Même pour des systèmes les plus simples, le nombre de scénarios d'exécution peut être est soit infini soit très grand. Par conséquent, la simulation ou les tests exhaustifs ne peuvent pas être utilisés pour vérifier le comportement temporel de ces systèmes. Aussi, des techniques d'analyse formelle sont nécessaires pour garantir que les systèmes conçus sont, par construction, temporellement correct et prédictible [42, 47]. Au fil du temps, les applications temps réel sont devenues de plus en plus sophistiquées et complexes dans leurs comportements et leurs interactions. Ces exigences applicatives ont permis le développement d'architectures multi-cœur, architectures fortement répandues dans les systèmes commerciaux actuels. Bien que des recherches importantes aient été menées sur la conception de systémes temps-réel au cours des derni‘eres décennies, l'émergence des architectures multi-cœurs a entrainé de nouveaux défis notamment pour la communauté de recherche sur les systèmes temps réel. Ces défis peuvent être classés en trois grandes catégories: la conception d'architecture multiprocesseur, l'ordonnancement multiprocesseur, et la gestion de l'énergie dans les systèmes multiprocesseurs.

Comme les architectures multiprocesseurs sont déjà largement utilisées, il devient de plus en plus évident que les futurs systèmes temps réel seront déployés sur ces architectures. Les architectures multiprocesseurs possèdent de nouvelles fonctionnalités qui doivent être prises en considération. Par exemple, les programmes applicatifs qui s'exécutent sur différents cœurs partagent en général des ressources (grain fin), comme les caches unifiés, les réseaux d'interconnexion et la bande passante de la mémoire, ce qui rend les pratiques classiques employées lors de la conception non applicables à ces systèmes multi-cœurs. Aussi, les

architectures multi-cœurs présentent plus de difficultés lors de leur conception, leur analyse et leur mise en œuvre.

Un autre défi pour les systèmes temps réel réside dans le problème de l'ordonnancement. Le problème de l'ordonnancement temps réel pour des systèmes multiprocesseurs est très différent et plus complexe que pour des systèmes mono-processeur. Les algorithmes d'ordonnancement développés dans un contexte mono-processeur ne peuvent pas être appliqués sur les systèmes multiprocesseurs sans perte d'optimalité. Un algorithme d'ordonnancement est dit optimal s'il est capable d'ordonnancer n'importe quel système de tâches qualifié d'ordonnancable [105]. Un système de tâche est dit ordonnancable si il existe un ordonnancement tel que les échéances temporelles des jobs soient respectées, et ceci pour toutes les séquences de jobs qui peuvent être générées à partir du système de tâches initial. L'optimalité des algorithmes d'ordonnancement est une question critique lors de la conception de systèmes temps réel multiprocesseurs compte tenu du fait que la sous-utilisation des ressources n'est pas souhaitable. Les algorithmes d'ordonnancement multiprocesseur utilisent soit une approche d'ordonnancement partitionné soit global (ou hybride des deux). Pour les ordonnancements partitionnés, les tâches sont attribuées de manière statique aux processeurs et ordonnancée sur chaque processeur en utilisant des algorithmes d'ordonnancement monoprocesseur. Ce type d'ordonnancement présente un surcoût faible d'un point de vue de l'ordonnancement. Toutefois, la gestion des ressources globalement partagées comme une mémoire principale et les caches peut devenir très difficile lors du partitionnement, précisément parce que chaque processeur construit son ordonnancement de façon indépendante. En outre, l'affectation des tâches aux processeurs est souvent considérée comme la résolution d'un problème de *bin-packing*: sur un système de m-processeurs, chaque tâche avec une taille égale à son utilisation doit être placée dans un des m *bins* de taille un et qui représente un processeur. Le *bin-packing* est considéré comme un problème NP-complet [60]. Dans les algorithmes d'ordonnancement global, tous les processeurs sélectionnent les jobs â ordonnancer dans une file d'exécution unique. En conséquence, les jobs peuvent migrer entre les processeurs, et les conflits liés aux structures de données partagées sont probables. Jusqu'â présent, aucun algorithme optimal d'ordonnancement global multiprocesseur n'existait avant la proposition de PFair et ses algorithmes heuristiques [13, 106]. Bien que quelques algorithmes récemment proposés sont connus pour être optimaux [13, 106, 77, 28], la théorie de l'ordonnancement multiprocesseur pose de nombreux problèmes fondamentaux qui sont toujours non résolus.

La complexité croissante des applications temps réel qui sont ordonnancées sur des architectures multiprocesseurs, issues aussi bien du multimédia, des télécommunications ou des applications aérospatiales, pose un autre grand problème, celui de la consommation énergétique de ces équipements électroniques qui a augmenté de façon exponentielle. La densité de puissance au sein des microprocesseurs a presque double tous les trois ans [103, 56]. L'augmentation de la consommation engendre deux types de difficultés: la consommation d'énergie intrinsèque et l'élévation de la température dans les circuits. Comme l'énergie correspond à l'intégration de la puissance en fonction du temps, fournir l'énergie nécessaire peut devenir complexe et onéreux, voir même techniquement impossible. Cette difficulté est d'autant plus problématique dans les systèmes portables alimentés par des batteries et elle deviendra encore plus critique dans le futur car la capacité des batteries augmente à un rythme beaucoup plus lent que celui de la puissance consommée.

L'énergie consommée dans les systèmes électroniques est en grande partie transformée en chaleur. Les plates-formes de traitement se tournent actuellement vers des architectures 3D [30, 104], pour lesquelles la distribution de la température et la consommation d'énergie deviennent des contraintes de conception de premier ordre. Par conséquent, les systèmes

temps réel complexes doivent impérativement réduire leur consommation d'énergie tout en garantissant le respect des contraintes temporelles. La gestion de l'énergie dans les systèmes temps réel a été abordée d'un point de vue matériel et/ou logiciel. De nombreuses approches logicielles, en particulier des approches d'ordonnancement, basées sur l'ajustement conjoint en tension et en fréquence (Dynamic Voltage and frequency scaling: DVFS) et la gestion des modes repos (Dynamic Power Management: DPM) ont été proposées par la communauté de recherche temps réel au cours de ces dernières années. Malheureusement, leur flexibilité va souvent de pair avec la complexité de la solution, et avec le risque que les délais ne soient parfois pas respectés. Comme la demande en terme de puissance de calcul des systèmes temps réel embarqués continue de croître, le développement d'approches efficientes et transparentes à l'utilisateur de gestion de l'énergie va devenir de plus en plus important afin de minimiser la consommation d'énergie, de prolonger la vie de la batterie, et de réduire les effets thermiques. L'efficacité énergétique et la programmation des systèmes temps-réel sont des problèmes étroitement liés, qui doivent être traités ensemble pour obtenir de meilleurs résultats. En exploitant les paramètres caractéristiques des tâches des applications temps réel, l'efficacité énergétique des algorithmes d'ordonnancement et la qualité de service relative à ces applications peuvent être sensiblement améliorée. Dans la suite de ce document, nous établissons les objectifs de la thèse.

Objectifs de la thèse: L'objectif de cette thèse est d'améliorer, à travers la problématique de l'ordonnancement, l'efficacité énergétique des systèmes temps réel qui peuvent être prouvés (démontrés) prédictible et temporellement correct et ceci dans un contexte multiprocesseurs. Les solutions proposées doivent être flexibles afin de s'adapter aux exigences variables des systèmes, de faible complexité, et les plus efficaces possibles. Pour atteindre cet objectif, il est nécessaire que les systèmes temps réel fonctionnant sur batterie puissent toujours répondre aux contraintes temporelles tout en: minimisant la consommation d'énergie, prolongeant la vie de la batterie, et réduisant des effets thermiques.

Aussi, cette thèse propose des solutions au sein des algorithmes d'ordonnancement en intégrant une gestion de l'énergie pour des applications temps-réel complexes qui seront ordonnancées sur des architectures multiprocesseurs. Dans le paragraphe 1.2, nous donnons un aperçu de chaque contribution technique présentée tout au long de cette thèse. Les fondamentaux relatifs aux systèmes temps réel et à la gestion de l'énergie lors de l'ordonnancement de ces systèmes sont fournis au chapitre 2. Il faut noter que l'état de l'art relatif aux différentes contributions se trouve en début de chaque chapitre. Toutefois, les travaux de recherche connexes sont également référencés dans le document au moment opportun.

1.2 Contributions

L'efficacité énergétique dans les systèmes temps réel est un problème d'optimisation multifacettes. Par exemple, l'optimisation énergétique peut être réalisée à la fois à des niveaux matériel et logiciel, pendant la conception du système et lors de l'ordonnancement pendant l'exécution de tâches de l'application. A la fois le matériel et le logiciel sont concernés et peuvent jouer un rôle important dans la consommation énergétique de l'ensemble du système. Dans ce mémoire, nous nous concentrons sur les aspects logiciels, plus particulièrement sur la gestion de l'énergie lors de l'ordonnancement dans les systèmes temps réel. Nous développons des techniques novatrices de gestion de la puissance et de l'énergie tout en prenant en compte les fonctionnalités offertes par les architectures existantes et futures de plateformes. Dans la suite, nous discutons des différentes contributions présentées dans chaque chapitre de cette thèse.

Chapitre 3. Dans ce chapitre, nous présentons notre première contribution qui est un algorithme d'ordonnancement multiprocesseur, appelé 2L-HiSA (Two-Level Hierarchical Scheduling Algorithm) algorithme d'ordonnancement hiérarchique à deux niveaux. Cet algorithme se situe dans la catégorie des ordonnancements à migration restreinte. L'algorithme d'ordonnancement EDF possède la plus faible complexité d'exécution parmi les algorithmes à priorité fixe au niveau des jobs pour une architecture multiprocesseur. Cependant, EDF souffre de sous-optimalité dans les systèmes multiprocesseurs. 2L-HiSA adresse la sous-optimalité d'EDF comme algorithme d'ordonnancement global et décompose le problème en une hiérarchie d'ordonnanceurs à deux niveaux. Nous avons veillé à ce que les propriétés intrinsèques de base de l'algorithme d'ordonnancement EDF mono-processeur apparaissent dans la hiérarchie d'ordonnanceurs à deux niveaux tant à l'ordonnanceur haut niveau qu'à l'ordonnanceur au niveau local. 2L-HiSA alloue (partitionne) les tâches statiquement aux processeurs en utilisant l'approche du *bin-packing*, tel que l'ordonnancabilité des tâches partitionnées sur un processeur donné ne soit pas violé. Les tâches qui ne peuvent être partitionnées sur un processeur de la plateforme sont appelées tâches migrantes ou globales. En outre, des clusters de processeurs identiques sont construits de telle sorte que, par cluster, l'équivalent de la puissance de calcul non utilisée fragmentée soit disponible par au plus un processeur. Nous montrons que 2L-HiSA améliore les limites de l'ordonnancabilité d'EDF pour des systèmes multiprocesseurs et il est optimal pour des tâches temps réel dur si un sous-ensemble des tâches peut être partitionné de telle sorte que la sous-utilisation par cluster des processeurs reste inférieure ou égale â l'équivalent d'un processeur. Le partitionnement des tâches sur les processeurs réduit le surcout dû à l'ordonnancement tel que les changements de contexte, les préemptions et les migrations, ce qui contribue à réduire la consommation énergétique globale. Cependant, la nature NP-difficile du problème du partitionnement [60], peut souvent être un facteur limitant. En utilisant des clusters de processeurs au lieu de considérer des processeurs individuels, 2L-HiSA atténue les limitations du *bin-packing* en augmentant effectivement la taille des *bin* par rapport à la taille des objets. Avec un cluster de processeurs, il est beaucoup plus facile d'obtenir la puissance inutilisée de traitement par cluster inférieure ou égale à un seul processeur. Les résultats de la simulation obtenus montrent tout l'intérêt de cette proposition.

Chapitre 4. La seconde contribution, présentée dans ce chapitre, est une technique de gestion dynamique de la puissance pour les systèmes temps réel multiprocesseurs, appelée *Assertive Dynamic Power Management (AsDPM)*. Cette technique travaille en conjonction avec l'algorithme d'ordonnancement EDF global. Il s'agit d'une technique de contrôle d'admission des tâches temps réel qui décide à quel moment exactement une tâche prête peut être exécutée. Sans ce contrôle d'admission, toutes les tâches prêtes seraient exécutées dès qu'il y a suffisamment de ressources de calcul (processeurs) disponibles dans le système, conduisant à une faible possibilité de mettre des processeurs dans un état faible consommation. La technique AsDPM diffère de la technique DPM existante dans la façon dont il exploite les intervalles de temps d'inactivité. Les techniques classiques de DPM peuvent exploiter des intervalles d'inactivité lorsqu'ils se produisent sur un processeur, c'est-à-dire, une fois que l'intervalle temporel d'inactivité est détecté. Lors de la détection des intervalles temporels d'inactivité, ces techniques décident de passer le processeur cible dans un mode faible consommation. Par contre, la technique AsDPM extrait de manière agressive les intervalles d'inactivité de certain processeurs et les groupent sur d'autres processeurs de la plateforme pour allonger la durée d'inactivité. Le passage des processeurs dans un mode faible consommation approprié revient alors à comparer la durée des intervalles temporels d'inactivité avec les temps minimumde pause du processeur cible correspondant à un

bilan énergétique. Bien que AsDPM soit une technique dynamique en ligne de gestion de l'énergie, son principe de fonctionnement peut être utilisé pour déterminer les configurations statiques architecturales optimisées (le nombre de processeurs et le couple tension-fréquence correspondant nécessaires pour répondre aux contraintes temps réel dans le pire des cas avec une consommation d'énergie minimale) pour l'application cible par le biais de simulations. Nous démontrons dans ce chapitre l'utilisation de la technique AsDPM pour l'optimisation statique et dynamique de l'énergie.

Chapitre 5. Ce chapitre présente la troisième contribution, qui est une technique dynamique d'ajustement conjoint de la tension et de la fréquence inter-tâches pour les systèmes multiprocesseurs temps réel, appelée *Deterministic Stretch-to-Fit (DSF)*. La technique DSF est principalement destinée aux systèmes multiprocesseurs. Il est également possible de l'appliquer aux systèmes mono-processeurs, dans ce cas elle devient alors triviale en raison de l'absence de migration des tâches. DSF est composé de trois algorithmes: l'algorithme *Dynamic Slack Reclamation (DSR)*, *Online Speculative speed adjustment Mechanism (OSM)*, et l'algorithme *m-Tasks Extension (m-TE)*. L'algorithme DSR est l'algorithme de gestion des intervalles de temps inutilisés qui attribue l'intervalle de temps inutilisé par une tâche précédente à la tâche prête suivante ayant la priorité locale la plus élevées s'exécutant sur le même processeur. Lors de l'utilisation de l'algorithme DSR, les intervalles de temps inutilisés ne sont pas partagés avec d'autres processeurs du système. Un intervalle temporel inutilisé par une tâche est entièrement consommé par le processeur auquel la tâche a été allouée. Cette attribution gloutonne des intervalles permet à l'algorithme DSR de calculer un facteur de ralentissement plus important afin de réduire à la fois la tension et la fréquence de fonctionnement pour une tâche unique, permettant au final une augmentation des économies d'énergie. Les algorithmes OSM et *m-TE* sont des extensions de l'algorithme de DSR. L'algorithme OSM est un mécanisme en ligne adaptatif et spéculatif d'ajustement de la vitesse de fonctionnement qui prévoit les dates de fin d'exécution des tâches et effectue un ralentissement agressif de la vitesse du processeur. En plus de économie d'énergie réalisée par l'algorithme de DSR, OSM contribue également à éviter des changements la fréquence de fonctionnement et de la tension d'alimentation, ce qui réduites consommation d'énergie fournisant une augmentation de la durée de vie des batteries notamment pour les systèmes embarqués portables. L'algorithme *m-TE* étend la technique *One-Task Extension (OTE)* pour des systèmes mono-processeur aux systèmes multiprocesseurs. La technique de DSF est générique en ce sens que, si pour une application cible temps réel donnée, un ordonnancement existe basé sur le pire cas de la charge de travail (optimal ou non-optimal) en utilisant un algorithme d'ordonnancement global, alors le même ordonnancement peut être reproduit (en utilisant la charge de travail réelle) avec une consommation de puissance et d'énergie moindre. Ainsi, DSF peut travailler en collaboration avec divers algorithmes d'ordonnancement. DSF est basé sur le principe du suivre de l'exécution canonique des tâches au moment de l'exécution, c'est-à-dire, l'ordonnancement calculé hors ligne (ou statique) dans lequel le temps d'exécution au pire cas de tous les jobs des tâches est considéré. Une trace d'exécution de toutes les tâches de l'ordonnancement optimale statique doit être conservée afin de le suivre lors de l'exécution [10]. Cependant, l'établissement et la mémorisation de cet ordonnancement canonique dans son intégralité est impossible dans le cas des systèmes multiprocesseurs en raison du fait que l'affectation des tâches préemptives et migrantes aux processeurs n'est pas connue a priori. Par conséquent, nous proposons un schéma pour produire un ordonnancement en ligne canonique en avance sur l'ordonnancement pratique, qui imite l'exécution canonique des tâches seulement pour les m-tâches futures. Cette caractéristique réduit les surcoûts d'exécution liés à l'ordonnanceur

ce qui fait de DSF une technique adaptative.

Chapter 6. Alors que de nouvelles techniques de gestion de l'énergie sont encore en développement pour répondre à des conditions spécifiques d'exploitation, des recherches plus récentes montrent que l'efficacité des deux techniques DPM et DVFS est très variable lorsque les conditions de fonctionnement changent [37, 20]. Ainsi, aucune politique ne s'inscrit parfaitement dans la totalité ou la plupart des conditions de fonctionnement. En réponse à ce probléme une quatrième et dernière contribution a été étudiée. Aussi, nous proposons, dans ce chapitre, un schéma générique de gestion de l'énergie et de la puissance pour des systèmes multiprocesseurs temps réel appeléeÊ*Hybrid Power Management (HyPowMan).* Ce schéma est utilisé comme une entité de contrôle qui au lieu de concevoir une ou des politiques de gestion de la puissance et de l'énergie pour des conditions opératoires spécifiques, exploite un ensemble de politiques existantes. Chaque politique présente dans l'ensemble des politiques utilisée, quand elle fonctionne seule, doit pouvoir fournir les garanties des échéances des tâches. En cours d'exécution, la politique la plus performante pour une charge de travail donnée est adoptée par le schéma HyPowMan grâce à un algorithme de type *machine-learning.* Ce schéma peut améliorer la capacité des systèmes embarqués portables pour s'adapter aux variations de la charge de travail (et de configuration des plateformes) en travaillant avec un plus large ensemble de conditions opératoires et fournir des performances globales avec des gains énergétiques qui soient meilleures que ceux que pourrait offrir une politique unique d'optimisation.

Chapter 7. Dans ce chapitre, nous présentons les conclusions générales et les remarques sur nos contributions et nos résultats. De plus nous discuterons des perspectives de recherches liées à ce travail de thèse.

Annexes. Deux annexes sont fournies dans ce mémoire de thèse. L'annexe A présente les détails fonctionnels relatifs à l'outil de simulation STORM (Simulation TOol for Realtime Multiprocessor scheduling) [108] que nous avons utilisé dans nos simulations tout au long de ce mémoire. L'annexe B donne des résultats de simulation additionnels relatifs au chapitre 6.

1.3 Résumé

Suite à l'évolution récente de la complexité et du niveau de sophistication des applications temps réel et des plateformes multiprocesseurs, la recherche sur les systèmes temps réel est confrontée à de nombreux défis émergents. Un de ces défis auquel la communauté de recherche temps réel est confrontée est de réduire la consommation d'énergie et de puissance de ces systèmes, tout en assurant que les contraintes temporelles seront respectées. Comme les exigences en puissance de calcul des systèmes temps-réel continuent de croître, il apparaît de plus en plus indispensable de développer des approches de minimisation de la consommation d'énergie afin de prolonger la durée de vie de la batterie, et de réduire les effets thermiques. L'efficacité en termes de puissance et d'énergie et l'ordonnancement de systèmes temps réel sont des problèmes étroitement liés, qui devraient être traités ensemble afin d'obtenir de meilleurs résultats. Ces différentes raisons ont motivé ce travail de thèse qui tente d'apporter des solutions aux problèmes de l'efficacité énergétique et de l'ordonnancement des systèmes temps-réel multiprocesseurs. Cette thèse propose de nouvelles approches pour la gestion énergétique au travers du paradigme de l'ordonnancement faible consommation pour des applications temps réel souple et dur sur

des plateformes multiprocesseurs. Les plateformes considérées sont de type *SMP (Symmetric shared-memory MultiProcessor)*. Nous mentrons qu'en exploitant les paramètres caractéristiques des tâches des applications temps réel, l'efficacité énergétique au travers des algorithmes d'ordonnancement à qualité de service constante peut être améliorée. Dans le reste de ce document, les différentes contributions de ce travail de thèse sont détaillées.

Chapter 2

Conclusions et Perspectives

Les systèmes embarqués temps réel sont devenus omniprésents dans notre vie, que ce soit les téléphones cellulaires portatifs, les appareils ménagers ou des systèmes plus sophistiqués comme les systèmes de traitement de signal, les satellites, et les applications militaires. Comme l'utilisation de ces systèmes temps réel est diversifiée, les travaux de recherche relatifs à ces systèmes sont confrontés à de nombreux défis émergents. Actuellement, les applications temps réel sont devenues sophistiquées et complexes dans leurs comportement et leurs interactions. Ce phénomène a permis l'émergence des architectures multiprocesseurs. Le développement de ces architectures multiprocesseurs dans le secteur commercial, est aussi du au fait que ces architectures sont considérées comme une solution à la barrière thermique rencontrée dans les architectures mono-processeur. Il est devenu clair qu'à l'avenir, la plupart des applications temps réel complexes seront déployées sur des plateformes multiprocesseurs. Inévitablement, cette évolution lors de la conception de ces nouvelles architectures ainsi que la complexité des applications temps réel a réactualisé certains défis actuels qui impactent (ou intéressent fortement) fortement la communauté de recherche temps réel.

Un des problèmes majeurs, c'est que les algorithmes d'ordonnancement optimal monoprocesseur ne peuvent être appliqués sur les systèmes multiprocesseurs sans perte d'optimalité. Ainsi, la communauté de recherche en temps réel doit élaborer des stratégies d'ordonnancement alternatives pour intégrer des aspects relatifs aux systèmes multiprocesseurs. Un autre challenge auxquels sont confrontés les systèmes temps réel est de réduire la consommation d'énergie et de puissance tout en s'assurant que les contraintes de temps seront respectées. Comme les exigences en termes de puissance de calcul des systèmes temps réel embarqués ne cessent de croître, il est nécessaire de développer des approches efficientes de gestion de l'énergie afin de minimiser la consommation d'énergie, d'étendre la durée de vie de la batterie, et de réduire les effets thermiques. La consommation d'énergie et de puissance dans les systèmes temps réel revêt une importance capitale depuis ces dernières années et de nombreuses approches logiciel, aussi bien statique que dynamique, permettant de réduire la consommation ont été proposées. Ces solutions logicielles, notamment les politiques d'ordonnancement conscientes du problème énergétique comme l'adaptation dynamique de la tension et de la fréquence (DVFS) et la gestion des modes faible consommation (ou mode repos)(DPM) ont vu le jour. En outre, des techniques efficaces énergétiquement à l'échelle du système c'est-à-dire qui agissent sur des sous-systèmes comme la mémoire et le réseau d'interconnexion ont aussi été proposées. Malheureusement, leur flexibilité va souvent de pair avec la complexité de la solution. Les travaux présentés dans cette thèse visent à améliorer la gestion de l'énergie et de la puissance dans les systèmes embarqués temps réel en proposant de nouvelles stratégies flexibles et efficientes au sein de l'ordonnancement portant alors le qualificatif *energy-aware*. Les techniques proposées dans cette thèse augmentent le spectre des conditions de fonctionnement dans lequel, les applications temps réel peuvent être ordonnancées de manière à réduire l'énergie consommée tout en conservant leurs garanties temporelles pour des plateformes multiprocesseurs. Nous avons choisi de diviser ce chapitre en deux parties. La première résume les différentes contributions et les résultats significatifs obtenus. La deuxième partie présente des améliorations relatives et

des perspectives de recherche.

2.1 Résumé des Contributions et Résultats

Dans ce paragraphe, nous résumons les contributions de cette thèse.

Algorithme d'ordonnancement hiérarchique à 2 niveaux (Two-Level Hierarchical Scheduling Algorithm 2L-HiSA). Les stratégies d'ordonnancement à migration restreinte fournissent un bon compromis entre la migration complète et les approches partitionnées [24, 63, 62]. Ces stratégies sont suffisamment flexibles pour permettre aux tâches dynamiques de se joindre au système lors de l'exécution, et elles n'engendrent pas de surcoût importante lié à la migration par rapport à des stratégies avec migration non restreinte. Dans cette thèse, et plus spécifiquement dans le chapitre 3, nous avons présenté un algorithme d'ordonnancement multiprocesseur hiérarchique à deux niveaux (appelé 2L-HiSA), qui tombe dans la catégorie des ordonnancements avec migration restreinte. L'algorithme d'ordonnancement EDF possède la plus faible complexité d'exécution parmi les algorithmes à priorité fixe au niveau des jobs lors l'ordonnancement des tâches sur une architecture multiprocesseur. Toutefois, EDF souffre de sous-optimalité dans les systèmes multiprocesseurs. 2L-HiSA exploite la sous-optimalité d'EDF comme algorithme d'ordonnancement global et divise le problème en une hiérarchie à deux niveaux d'ordonnanceurs. 2L-HiSA utilise plusieurs instances de l'algorithme optimal d'ordonnancement EDF mono-processeur par une hiérarchie à deux niveaux : une instance d'EDF au niveau de l'ordonnancement haut niveau et une instance d'ordonnancement au niveau local pour chaque processeur de la plateforme. En outre, 2L-HiSA comporte deux phases: 1) la phase de partitionnement des tâches, dans laquelle, chaque tâche non-migrante est attribuée à un processeur spécifique en suivant l'approche de *bin-packing*, 2) la phase de regroupement processeur, dans laquelle, les processeurs sont regroupés en fonction de leurs caractéristiques de charge de travail. 2L-HiSA partitionnent statiquement les tâches de l'application sur les processeurs en utilisant l'approche de bin-packing tant que l'ordonnancabilit des tâches partitionnées sur un processeur particulier est respctée. Les tâches qui ne peuvent être partitionnées sur aucun processeur de la plateforme sont considérées migrantes ou globales. De plus, la construction des clusters de processeurs identiques fait que, par cluster, la puissance de calcul fragmentée non utilisée équivalente au plus un processeur est disponible. Nous montrons que 2L-HiSA améliore les limites de l'ordonnanabilité d'EDF pour les systèmes multiprocesseurs et est optimal pour des tâches temps réel dur si un sous-ensemble de tâches partitionnées peut être de telle sorte que la sous-utilisation par cluster de processeurs reste inférieure ou égale à la charge d'un processeur. Le partitionnement des tâches sur les processeurs réduit le surcoût lié à l'ordonnancement comme par exemple le changement de contexte, les préemptions et les migrations. Ce fait contribue à réduire la consommation d'énergie globale. Cependant, la difficulté NP-complet du problème du partitionnement [60], est souvent un facteur limitant. En utilisant des clusters de processeurs plutôt que des processeurs individuels, 2L-HiSA atténue les limites du bin-packing en augmentant les tailles des *bin* par rapport à la taille des objets. Avec un cluster de processeurs, il est beaucoup plus facile d'extraire la puissance inutilisée de traitement par cluster, qui est inférieure ou égale à un seul processeur. Des résultats de simulation ont permis de vérifier la validité de cette proposition. Nous avons montré que les stratégies efficaces en termes d'énergie et de puissance comme le DVFS et / ou DPM peuvent être utilisées en conjonction avec 2L-HiSA pour améliorer les gains énergétiques. De plus, nous avons montré que la préemption des tâches et les surcoûts liés à la

migration sont beaucoup moins importants pour 2L-HiSA par rapport à PFair et ASEDZL, qui sont des algorithmes d'ordonnancement multiprocesseur optimal.

La Technique AsDPM (Assertive Dynamic Power Management). La seconde contribution de cette thèse, qui est présentée dans le chapitre 4, fut de proposer une technique dynamique de gestion de l'énergie pour les systèmes temps-réel multiprocesseur. Les techniques DPM permettent d'obtenir des gains énergétiques dans les systèmes électroniques embarqués en changeant le profil de consommation du système c'est-à-dire en plaçant les composants dans des modes faible consommation adéquats pour respecter les exigences applicatives. La technique proposée est appelée *Assertive Dynamic Power ManagementÊ* (AsDPM) en raison de sa capacité d'extraction et de cumul des intervalles temporels d'inactivité, ce qui n'est pas le cas dans les techniques classiques DPM. La technique AsDPM travaille sous le contrôle des algorithmes d'ordonnancement, global EDF et global LLF. Il s'agit d'une technique de contrôle d'admission des tâches temps réel qui rend l'ensemble des tâches efficaces énergétiquement en décidant quand exactement une tâche prête doit être exécutée. Sans ce contrôle d'admission, toutes les tâches seraient exécutées lorsque un nombre suffisant de ressources seraient disponibles dans le système, menant à de faible possibilité de faire transiter les processeurs en mode faible consommation. La technique AsDPM se différencie des autres techniques DPM existantes dans le sens où elle exploite les intervalles de temps d'inactivité. Une technique DPM conventionnelle peut exploiter une période d'inactivité seulement lorsqu'elle apparait sur un processeur c'est-à-dire lorsque cet intervalle d'inactivité est détecté. Une fois les intervalles d'inactivité détectés, ces techniques décident de transiter en mode faible consommation les processeurs cibles. La technique AsDPM, d'un autre coté, extrait tous les intervalles d'inactivité des processeurs et les groupent sur un ou plusieurs processeurs de la plateforme pour allonger la durée de ces temps d'inactivité. Le fait de placer les processeurs dans un mode repos adapté, revient alors à comparer la longueur des intervalles de repos avec le seuil de rentabilité du mode repos concerné du processeur cible. Bien qu'AsDPM soit une technique dynamique en ligne de gestion de la consommation, son principe de travail peut être utilisé pour déterminer des configurations architecturales statiques optimales (c'est à dire, nombre de processeurs, et leur couple tension/fréquence qui sont exigées pour satisfaire les contraintes temps réel au cas pire avec le minimum d'énergie) pour des applications cibles au travers de simulations. Dans cette thèse, nous avons démontré, l'utilisation d'AsDPM pour déterminer en premier lieu les configurations statiques optimales et par la suite, comme une technique en ligne qui peut encore réduire l'efficacité énergétique des configurations déjà statiquement optimisées.

La Technique DSF (Deterministic Stretch-to-Fit DVFS). L'ajustement conjoint en tension et en fréquence est une des techniques dont le but est de changer le profile de consommation des systèmes embarqués temps réel. Ceci est du au fait que la consommation est une fonction quadratique de la tension d'alimentation des processeurs [48]. Les applications temps réel offrent potentiellement des variations dans leur temps d'exécution et par conséquent finissent plutôt que le pire temps d'exécution estimé. Les techniques DVFS temps réels exploitent ces variations au travers de la charge de travail réelle en ajustant dynamiquement la tension et la fréquence des processeurs afin de réduire la puissance consommée. Dans cette thèse, chapitre 5, nous proposons une technique dynamique intertâche d'ajustement conjoint en tension et en fréquence pour des systèmes multiprocesseurs temps réel appelé:*Deterministic Stretch-to-Fit (DSF)*. La technique DSF est principalement destinée aux systèmes multiprocesseurs. Il est également possible de l'appliquer aux systèmes mono-processeurs, dans ce cas elle devient alors triviale en raison de l'absence de

migration des tâches. DSF est composé de trois algorithmes: un algorithme *Dynamic Slack Reclamation (DSR)*, algorithme *Online Speculative speed adjustment Mechanism (OSM)*, et l'algorithme *m-Tasks Extension (m-TE)*. L'algorithme DSR est l'algorithme de gestion des intervalles de temps inutilisés qui attribue l'intervalle de temps inutilisé par une tâche précédente à la tâche prête suivante ayant la priorité locale la plus élevée s'exécutant sur le même processeur. Lors de l'utilisation de l'algorithme DSR, les intervalles de temps inutilisés ne sont pas partagés avec d'autres processeurs du système. Un intervalle temporel inutilisé par une tâche est entièrement consommé par le processeur auquel la tâche a été allouée. Cette attribution gloutonne des intervalles permet à l'algorithme DSR de calculer un facteur de ralentissement plus important afin de réduire à la fois la tension et la fréquence de fonctionnement pour une tâche unique, permettant au final une augmentation des économies d'énergie. Les algorithmes OSM et *m-TE* sont des extensions de l'algorithme de DSR. L'algorithme OSM est un mécanisme en ligne adaptatif et spéculatif d'ajustement de la vitesse de fonctionnement qui prévoit les dates de fin d'exécution des tâches et effectue un ralentissement agressif de la vitesse du processeur. En plus de l'économie d'énergie réalisée par l'algorithme de DSR, OSM contribue également à éviter des changements de la fréquence de fonctionnement et de la tension d'alimentation, ce qui réduits consommation d'énergie fournisant une augmentation de la durée de vie des batteries notamment pour les systèmes embarqués portables. L'algorithme *m-TE* étend la technique *One-Task Extension (OTE)* pour des systèmes mono-processeur aux systèmes multiprocesseurs. La technique de DSF est générique en ce sens que, si pour une application cible temps réel donnée, un ordonnancement existe basé sur le pire cas de la charge de travail (optimal ou non-optimal) en utilisant un algorithme d'ordonnancement global, alors le même ordonnancement peut être reproduit (en utilisant la charge de travail réelle) avec une consommation de puissance et d'énergie moindre. Ainsi, DSF peut travailler en collaboration avec divers algorithmes d'ordonnancement. DSF est basé sur le principe du suivre de l'exécution canonique des tâches au moment de l'exécution, c'est-à-dire, l'ordonnancement calculé hors ligne (ou statique) dans lequel le temps d'exécution au pire cas de tous les jobs des tâches est considéré. Une trace d'exécution de toutes les tâches de l'ordonnancement optimale statique doit être conservée afin de le suivre lors de l'exécution [10]. Cependant, l'établissement et la mémorisation de cet ordonnancement canonique dans son intégralité est impossible dans le cas des systèmes multiprocesseurs en raison du fait que l'affectation des tâches préemptives et migrantes aux processeurs n'est pas connue a priori. Par conséquent, nous proposons un schéma pour produire un ordonnancement en ligne canonique en avance sur l'ordonnancement pratique, qui imite l'exécution canonique des tâches seulement pour les m-tâches futures. Cette caractéristique réduit les surcoûts d'exécution liés à l'ordonnanceur ce qui fait de DSF une technique adaptative.

La technique HyPowMan (Hybrid Power Management). Les stratégies, à la fois, de gestion des modes faible consommation (DPM) et d'ajustement conjoint en tension et en fréquence (DVFS), sont très souvent utilisés par les politiques d'ordonnancement pour gérer la consommation d'énergie et de puissance dans les systèmes embarqués modernes. Alors que de nouvelles techniques de gestion de l'énergie sont encore en développement pour répondre à des conditions spécifiques d'exploitation, des recherches plus récentes montrent que l'efficacité des deux techniques DPM et DVFS est très variable lorsque les conditions de fonctionnement changent [37, 20]. Ainsi, aucune politique ne s'inscrit parfaitement dans la totalité ou la plupart des conditions de fonctionnement. En réponse à ce probléme une quatrième et dernière contribution a été étudiée. Aussi, nous proposons, dans ce chapitre, un schéma générique de gestion de l'énergie et de la puissance pour des systèmes

multiprocesseur temps réel appelé:*Hybrid Power Management (HyPowMan).* Ce schéma est utilisé comme une entité au plus haut niveau qui au lieu de concevoir des politiques de gestion de la puissance et de l'énergie pour des conditions opératoires spécifiques, exploite un ensemble de politiques existantes. Chaque politique présente dans l'ensemble des politiques sélectionnées, et quand fonctionnent seules, peut fournir les garanties des échéances. En cours d'exécution, la politique la plus performante pour une charge de travail donnée est adoptée par le schéma HyPowMan grâce à un algorithme de type *machine-learning.* Ce schéma peut améliorer la capacité des systèmes embarqués portables pour s'adapter aux variations de la charge de travail (et de configuration des plateformes) en travaillant avec un plus large ensemble de conditions opératoires et fournir des performances globales et des gains énergétiques qui soient meilleurs que ceux que pourrait offrir une politique unique.

2.2 Perspectives

Les systèmes temps réel ont des caractéristiques très complexes. La modification de l'un des aspects du système peut amener a se placer devant un problème très différent. Dans cette thèse, nous nous sommes concentrés principalement sur la réalisation des techniques de gestion de l'énergie dans les systèmes multiprocesseurs temps réel autour de l'ordonnancement. Nos contributions s'appuient des modéles spécifiques des tâches et des plate-formes de traitement qui sont présentées dans le chapitre 2. La généralisation de ces modèles augmente le nombre de systèmes possibles qui peuvent être utilisés par les applications temps réel. Par conséquent, des extensions de ce travail serait de généraliser le modèle de tâche et le modèle de traitement. Certaines modèles plus généralistes nécessitent d'à apporter des améliorations aux techniques proposées.

2.2.1 Modèle des tâches

Dépendance des tâches. Les travaux présentés dans cette thèse suppose un modèle des tâches indépendantes et préemptives, c'est-à-dire, l'exécution d'un job d'une tâche ne dépend pas du statut du job d'une autre tâche. Il y a beaucoup des systèmes pour lesquels l'hypothèse que les tâches son indépendantes n'est pas une hypothèse raisonnable. En outre, dans certains cas, la préemption des tâches ne peut pas être permise. Les tâches peuvent avoir des dépendances pour une variété de raisons. Les deux principaux types de dépendances sont liées aux contraintes du partage des ressources et de précédence. L'intégration des contraintes du partage des ressources et de précédences dans les résultats présentés tout au long de cette thèse serait un prolongement naturel. Toutefois, les dépendances entre les tâches peuvent provoquer des inversions de priorités, des situations où un job avec une priorité plus élevée est bloqué et un job avec une priorité faible s'exécute. Les inversions des priorités peuvent constituer une violation de nos hypothèses. Par exemple, dans le cas de l'algorithme d'ordonnancement 2L-HiSA, deux tâches avec des contraintes de précédence peuvent exiger d'être affectées de façon statique sur le même processeur, ajoutant ainsi une autre limitation au problème de partitionnement. Aussi, dans le cas de la technique As-DPM, le partage des ressources entre les tâches peut exiger qu'un processeur soit toujours en état *idle* qui aurait pu être autrement, mise dans l'état *repos* dès que la charge de travail est réduite.

Préemption des Tâches. La préemption de tâche est une hypothèse faite dans un grande nombre des systèmes temps réel, cependant, il y a des situations où les tâches ne

peuvent être préemptées. Le fait de ne pas autoriser la préemption des tâches peut provoquer des inversions de priorité. En outre, les systèmes non-préemptif peuvent aussi souffrir d'anomalies d'ordonnancement, c'est-à-dire, un système *faisable* peut dépasser les échéances si une ou plusieurs tâches sont retirées du système ou si l'exécution se terminent plustôt. La comptabilisation des problèmes d'inversions de priorité et des anomalies d'ordonnancement peut être un prolongement important du travail de la recherche présentée dans cette thèse. Cependant, nous n'avons pas considéré les systèmes non-préemptif. En outre, nous avons seulement examiné les systèmes faisable qui n'ont aucune anomalie d'ordonnancement.

Période de la Tâche. Une autre hypothèse courante dans les systèmes temps réel porte sur les modèles de tâche et de leur périodicité. Ce paramètre peut être interprété de trois manières distinctes, dont chacune conduit à un type de tâche bien défini. Selon l'interpretation donnée à la période de la tâche, les tâches peuvent être classées en trois catégories: modèle de tâche périodique, modèle de tâche sporadique, et modèle de tâche apériodique. Dans cette thèse, nous avons considéré un modèles périodique des tâches, c'est-à-dire, chaque tâche a un délai précise d'arrivée entre deux job successifs. L'extension de nos techniques proposées pour les modèles de tâches sporadique et apériodique serait une contribution intéressante car elle permettrait de considérer une plus large gamme d'applications temps réel.

2.2.2 Architectures de Plate-forme Cible

Cette thèse considère la plate-forme multiprocesseur homogène de type SMP (symétriric multiprocesseurs à mémoire partagée), composée de processeurs identiques. Il existe différentes façons pour généraliser le modèle de plate-forme de traitement. Certaines contributions de cette thèse, comme les techniques de AsDPM et DSF, ont été utilisés dans le projet national français PHERMA [86]. Ce projet propose un modèle hétérogène de plate-forme de traitement type *CMP (Chip Multi-Processing)* et qui est appelé *SCMP (Scalable Chip Multi-Processing)* qui prend en charge la migration dynamique et la préemption des tâches en utilisant la mémoire physiquement distribuée et logiquement partagée [120]. L'hétérogénité dans la plate-forme SCMP réside dans des clusters d'unités de traitement (c'est-à-dire, plusieurs clusters / groupes de processeurs identiques sont formés, toutefois, différents groupes peuvent contenir différents types d'unités de traitement). SCMP est une plate-forme qui se présente comme une alternative par rapport à la plate-forme SMP en prolongation de nos techniques proposées. En outre, les résultats présentés dans cette thèse considèrent surtout la consommation d'énergie au niveau des processeurs seuls et n'intégrent pas l'énergie consommée par d'autres sous-systèmes, comme par exemple, la mémoire et le réseau de d'interconnexion. Dans des travaux de recherche récents, il est démontré que les périphériques, les sous-systèmes de mémoire, les lecteurs de mémoire flash et les interfaces de réseau sans fil qui sont omniprésents dans les systémes embarqués modernes qui consomment aussi beaucoup d'énergie. Afin d'optimiser l'énergie de l'ensemble du système, des algorithmes d'ordonnancement qui permettent d'optimiser l'énergie au niveau des I/O doivent être développés pour les systèmes temps réel. L'extension des techniques proposées dans cette thèse au niveau système est une orientation possible. Par exemple, nous avons démontré au chapitre 4 que la technique AsDPM peut être utile pour les sous-systèmes comme la mémoire. La technique HyPowMan pout également intégrer la consommation d'énergie des périphériques et d'autres sous-systèmes tout en sélectionnant les experts appropriés.

Un autre aspect concernant la généralisation de la plate-forme matérielle réside dans

la mise de à l'échelle de la tension et la fréquence des processeurs qui prennent en charge plusieurs points de fonctionnement tension-fréquence. Dans cette thèse, nous avons estimé qu'il est possible de faire varier la tension et la fréquence de chaque processeur de façon indépendante et sur un spectre continu entre des bornes inférieures et supérieures définies. Cette considration est basée sur le fait que les processeurs qui sont capables de fonctionner sur un spectre de fréquence (plus ou moins) continue est en train de devenir une réalité [10]. Cependant, cette hypothèse peut être levée pour les processeurs offrant des points de fonctionnement discrets de fréquence tels que, si la vitesse optimale (calculée) du processeur n'est pas disponible sur un processeur, il doit être rapproché du niveau le plus proche discrèt et supérieur à la vitesse optimale pour respecter les contraintes temporelles. Cette conversion est simple pour les tâches qui satisfont la condition suffisante d'ordonnançabilité. Pour un ensemble de tâches qui satisfait à la condition nécessaire d'ordonnançabilité, la conversion de fréquence au discret amènerait une variation de l'utilisation au cours d'exécution, qui pourrait conduire à une perte d'échéance. Cet aspect nécessiterait des améliorations légères pour la technique de DSF, par exemple, qui est basée sur le principe que les tâches en cours d'éxecution doivent suivrent l'exécution canonique des tâches.

2.2.3 Les algorithmes d'ordonnancement

La plupart des contributions présentées dans cette thèse considèrent l'algorithme d'ordonnancement EDF pour l'ordonnancement des tâches temps réel sur une plate-forme multiprocesseurs. Même s'il existe plusieurs raisons pour lesquelles EDF est un algorithme d'ordonnancement raisonnable à considérer, il est pertinent de se demander si les contributions présentées dans ce mémoire peuvent aussi fonctionner en conjonction avec d'autres algorithmes d'ordonnancement. Nous avons discuté dans le chapitre 3 que l'algorithme EDF a une complexité d'exécution très faible pour l'ordonnancement des tâches sur des architectures multiprocesseurs, mais il souffre de sous-optimalité dans un contexte multiprocesseurs. Afin d'améliorer l'ordonnancement des systèmes liés avec EDF, nous avons proposé l'algorithme 2L-HiSA qui utilise plusieurs instances d'EDF, qui est optimal pour des systèmes mono-processeurs, dans une hiérarchie d'ordonnanceurs. Pour toutes les autres contributions de cette thèse, des extensions vers d'autres algorithmes serait une contribution précieuse. Nous discutons après des possibilités d'étendre nos techniques proposées à d'autres algorithmes d'ordonnancement multiprocesseur.

La Technique AsDPM. L'extension de la technique AsDPM à l'un des algorithmes d'ordonnancement multiprocesseur qui est basé sur le mécanisme d'ordonnancement *fluide* (par exemple, algorihme PFair [13] et algorithme LLREF [28]) n'est pas possible en raison de leur contradiction avec le principe de travail de la technique AsDPM. Dans les algorithmes d'ordonnancement basés sur le mécanisme des fluides, toutes les tâches prêtes doivent s'exécuter au prorata de leur utilisation respective (alors appelé *poids*). Dans la technique AsDPM, d'autre part, une tâche prête peut être différée (retarder) de l'exécution jusqu'à ce que sa *laxité anticipative* devienne négative (voir chapitre 4), ce qui est contraire aux principes de travail LLREF, PFair, et ses algorithmes heuristiques. Il est possible, cependant, d'appliquer cette technique à d'autres algorithmes comme LLF [71].

La Technique DSF. Il est possible d'appliquer les trois algorithmes, c'est-à-dire, DSR, OSM, et m-TE, présenté sous la technique DSF à d'autres algorithmes d'ordonnancement. Puisque la technique DSF fonctionne sur le principe de suivre l'exécution des tâches canoniques, par conséquent, il est possible d'intégrer la technique du DSF avec d'autres algo-

rithmes avec des modifications légères dans la méthode de calcul et l'attribution du *slack* dynamique.

La Technique HyPowMan. La technique HyPowMan elle-même n'intervient pas dans le processus décisionnel soit de l'algorithme d'ordonnancement soit d'expert sélectionné (stratégie de gestion de énergie). Ainsi, il est possible de coupler la technique HyPowMan avec d'autres algorithmes d'ordonnancement. Le seul facteur limitant peut être la complexité temporelle du système au global. Par exemple, l'association de la technique HyPowMan avec l'algorithme de PFair, qui est connu pour être complexe en raison de sa complexité d'ordonnancement, augmenterait considérablement la complexité (temps) du système.

2.2.4 Stratégie d'implementation

Dans cette thèse, nous nous sommes appuyés essentiellement sur des simulations pour l'évaluer et valider des techniques proposées. Notre motivation vient du fait que, tout d'abord, il est difficile de valider les résultats d'analyse, si les hypothèses sur les modèles de tâche et la plate-forme ne sont pas satisfaits. Satisfaire les hypothèses sur le modèle du système à l'aide de véritables plateformes n'est pas simple. En outre, dans les simulations, il est plus facile d'estimer les coûts associés à l'ordonnancement et d'évaluer la performance des techniques de gestion de l'énergie proposées en faisant varier les configurations système comme le type de processeur, le comportement du cache, les communications inter-processeurs, les dépendances entre les tâches, les variations du charge de travail, et les algorithmes d'ordonnancement. Nous avons utilisé un outil de simulation récent appelé STORM [108]. STORM est un outil de simulation basé sur Java pour les algorithmes d'ordonnancement multiprocesseur comprenant à la fois les paramètres de l'architecture matérielle et logicielle (voir l'annexe A pour plus de détails). Après avoir évalué la performance de nos techniques proposées en utilisant des simulations, nous travaillons à la mise en œuvre sur une véritable plate-forme. La plate-forme que nous utilisons pour le développement est ARM® ($ARM1176JZF-S$) [9]. Cette plate-forme offre la technique TrustZone® et une (IEM) gestion de l'énergie intelligente. En outre, nous sommes actuellement en train de mettre en œuvre les techniques AsDPM et DSF en utilisant la plate-forme virtuelle pour l'émulation de processeur appelée Rabbits (QEMU) [89]. Rabbits est un émulateur et virtualiseur. Il fournit un ensemble de modèles de plate-forme, ce qui lui permet d'exécuter une variété de systèmes d'exploitation. Il peut donc être considéré comme un moniteur de machine virtuelle hébergée.

2.2.5 Aspects Thermiques

Un des nouveaux défis auxquels sont confrontés les systèmes temps-réel est l'augmentation de la densité des transistors par puce en raison de la miniaturisation. L'autmentation de la densité sur puce et les exigences de calcul à haute vitesse conduisent à l'augmentation de la dissipation de chaleur dans les systèmes multiprocesseurs. Cette augmentation de la température provoque la création de *hot-spots (point chaud)*, ce qui réduit considérablement la durée de vie du composant. En outre, les concepteurs se concentrent désormais sur la fabrication d'architectures de plate-forme multi-core 3D [30, 104], afin de satisfaire aux exigences de calcul. Grâce à la miniaturisation, la surface disponible sur puce pour la dissipation thermique est réduite ce qui se traduit par une augmentation de la puissance dissipée et il faut aussi noté que le courant de fuite n'est pas réduit avec le même facteur. Les systèmes multiprocesseurs se comportent aussi comme plusieurs sources de chaleur ce

qui augmentent la probabilité des variations de température. L'énergie consommée dans le circuit est en grande partie transformée en chaleur. Aussi, les déséquilibres thermiques, ainsi que la consommation d'énergie sont devenus très important dans la conception des systèmes embarqués modernes. Le modèle du système considéré dans cette thèse contient des modèles de consommation de puissance et d'ènergie, qui sont relativement indépendants des aspects thermiques. Ce serait certainement une contribution intéressante et utile d'étendre nos techniques afin d'intégrer les aspects thermiques. Dans le chapitre 4, nous avons brièvement décrit le traitement des déséquilibres thermiques qui peuvent apparaître en raison de la technique AsDPM. De toute évidence, une étude plus précise et approfondie des aspects thermiques serait nécessaire.

2.3 Résumé

L'émergence de la technologie multi-core a apporté un changement de paradigme pour la recherche sur les systèmes temps réel embarqués. Les applications temps réel qui s'exécutent sur des plates-formes multiprocesseurs sont susceptibles d'être extrêmement diversifiés et caractérisées par le comportement du logiciel et des interactions complexes. La complexité des applications temps réel qui s'exécutent sur des plates-formes multiprocesseurs à remis au goût du jour, parmi les nombreux autres défis, le défi de l'optimisation de la consommation de puissance et d'énergie, tout en donnant l'assurance que les contraintes temps réel seront respectées. Cette thèse est basée sur l'hypothèse que l'efficacité énergétique et l'ordonnancement des systèmes temps-réel sont des problèmes étroitement liès, qui devraient être traités ensemble afin d'obtenir de meilleurs gaines. Les contributions proposées dans ce mémoire de thèse améliore, à travers l'ordonnancement, l'efficacité énergètique des systèmes temps réel qui peut être prouvé prédictibles et correctes dans son comportement temporele par rapport aux plateformes multiprocesseurs. Les solutions proposées sont flexibles au regard des exigences de système variable, moins complexes, et efficaces. La recherche à venir permettra d'éliminer certaines des hypothèses simplificatrices des modèles temps réel et d'architecture de la plateforme que nous avons utilisés. En outre, nous avons souligné que l'ensemble de l'efficacité énergétique et les effets thermiques devraient être pris en compte pour de meilleurs résultats au niveau système.

Liste des Publications

Refereed Journal Papers

1. **M. K. Bhatti**, C. Belleudy, M. Auguin, "Hybrid Power Management in Real-time Embedded Systems: An Interplay of DVFS and DPM Techniques", *In Springer's journal on Real-time Systems (RTS), special issue on Temperature/Energy Aware Real-Time Systems. DOI:* $10.1007/s11241 - 011 - 9116 - y$. *(To appear in early 2011).*
2. **M. K. Bhatti**, C. Belleudy, M. Auguin, "Two-level Hierarchical Scheduling Algorithm for Real-time Multiprocessor Systems", *In Journal of Software (JSW), Academy Publishers. (Accepted for publication).*

Refereed Conference Papers

1. **M. K. Bhatti**, M. Farooq, C. Belleudy, M. Auguin, O. Mbarek, "Assertive Dynamic Power Management (AsDPM) Strategy for Globally Scheduled RT Multiprocessor Systems", *In the proceedings of Power and Timing Modeling, Optimization and Simulation, PATMOS'*09*, and Integrated Circuit and System Design, chapter* 8*, Springer LNCS Vol.* 5953/2010*, ISBN* $978 - 3 - 642 - 11801 - 2$*, Pages* $116 - 126$*,* 2010.
2. **M. K. Bhatti**, C. Belleudy, M. Auguin, "Power Management in Real-time Embedded Systems through Online and Adaptive Interplay of DPM and DVFS Policies", *In the proceedings of International Conference on Embedded and Ubiquitous Computing, EUC'*10*, December* 2010*, Hong Kong, SAR, China.*
3. **M. K. Bhatti**, C. Belleudy, M. Auguin, "An Inter-Task Real-time DVFS Scheme for Multiprocessor Embedded Systems", *In the proceedings of International Conference on Design and Architectures for Signal and Image Processing, DASIP'*10*, October* 2010*, Edinburgh, UK.*
4. K. Ben Chehida, R. David, F. Thabet, **M. K. Bhatti**, M. Auguin, C. Belleudy, A.M. Déplanche, Y. Trinquet, R. Urunuela, , F. Broekaert, V.Seignole, A. M. Fouillart, "PHERMA, A global approach for system-level energy consumption optimization for Real-time heterogeneous MPSoC architectures", *In Proceedings of Low Voltage & Low Power Consumption Conference, FTFC'*09*,* 2009*, Neuchâtel, Switzerland.*
5. **M. K. Bhatt**i, C. Belleudy, M. Auguin, "A Framework for Offline Optimization of Energy Consumption in Real-time Multiprocessor System-on-Chip", *In the proceedings of IEEE International Conference on Electronics, Circuits, and Systems, ICECS'*09*, December* 2009*, Hammamet, Tunisia.*
6. **M. K. Bhatt**i, M. Farooq, C. Belleudy, M. Auguin, "Improving resource utilization under EDF-based mixed scheduling in multiprocessors real-time systems", *In the proceedings of IFIP/IEEE International Conference on Very Large Scale Integration, VLSI-SoC'*08*,* 2008*, Rhodes Island, Greece.*
7. M. Farooq, **M. K. Bhatti**, F. Muller, C. Belleudy, M. Auguin, " Precognitive DVFS: Minimizing Switching Points to Further Reduce the Energy Consumption", *In the proceedings of* 14*th IEEE Real-Time and Embedded Technology and Applications Symposium,* 2008*, St. Louis, MO, USA.*

Non-Refereed Workshop/Conference Papers

1. **M. K. Bhatti**, C. Belleudy, M. Auguin, "Controlling Energy Profile of Real-time Multiprocessor Systems by Anticipating Application's Workload at Runtime", *In the proceedings of SYMPosium on new Machine Architectures SympA'13, 2009, Toulouse, France.*
2. **M. K. Bhatti**, C. Belleudy, M. Auguin, "A hybrid scheduling algorithm for optimizing resource utilization & energy consumption in RT multiprocessor systems", *In the proceedings of 3rd National Symposium of Research Group on System-on-Chip, System-in-Package, GDR SoC – SiP'08, 2008, Paris, France.*
3. **M. K. Bhatti**, M. Farooq, C. Belleudy, M. Auguin, "Mixed Scheduling for Improved Resource Utilization and Energy Consumption in Real-time Multiprocessor Systems", *In the proceedings of Sophia Antipolis MicroElectronics, SAME'08, 2008, Sophia Antipolis, France.*

Appendix A

STORM: Simulation TOol for Real-time Multiprocessor Scheduling

In this dissertation, we have mainly relied on simulations for validating our proposed techniques. We use *STORM (Simulation TOol for Real-time Multiprocessor Scheduling)* simulator for this purpose. STORM is a free-ware Java-based simulation tool for multiprocessor scheduling algorithms developed by IRCCyN [116] and available under Creative Commons License. This tool has been initially designed and developed to evaluate the performance in term of energy efficiency for specific hardware and software architectures which are developed for French national project PHERMA (Parallel Heterogeneous Energy efficient real-time Multiprocessor Architecture) [86]. In this part of the dissertation, we provide some functional details of this simulation tool. These details are taken from [108, 119, 118].

Some of the main specifications of STORM are highlighted in the following.

- It is designed specifically for validating scheduling algorithms for multiprocessor architectures (composed of homogeneous or heterogeneous processors); single-processor architectures being the simplest case.
- It takes into account:
  - The features of hardware architecture: multi-core design, multiprocessor architecture with shared memory, distributed architecture with communication network, memory architecture (L1 and L2 caches, banked memory).
  - The features concerning energy consumption for the processors and memories, in particular the capabilities for DPM (Dynamic Power Management) and DVFS (Dynamic Voltage & Frequency Scaling).
- It is a flexible, portable, and open tool: *flexible* means the possibility to program and to add easily simulation entities (such as scheduling policies) through well-defined APIs; *portable* means the possibility to run it on various OS thanks to the Java programming language; and *open* means that the input/output data are formatted using xml.
- It is intended to provide a simulation language to drive experiments (statistical studies or domain explorations) via a simulation controller. Upstream from the simulator, this controller computes the inputs, runs the simulations, and downstream computes the required metrics from simulation results.

Figure A.1 illustrates that STORM simulator needs the specification of the studied real-time application as input –i.e., a set of tasks with execution requirements, that has to run on a multiprocessor hardware architecture. It is described in a XML input file in which specific tags and attributes have to be used, and where references to predefined

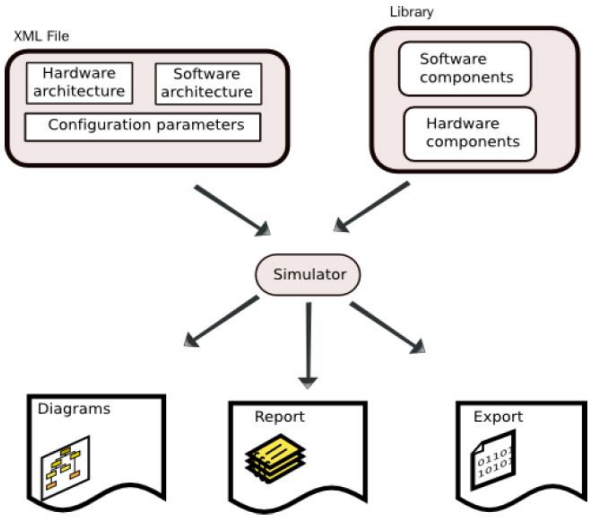


Figure A.1: STORM simulator input and output file system.

components are made. Thus, for a given application, the simulator is able to *play* its execution over a specified time interval while taking into account the requirements of tasks, the characteristics and functional conditions of hardware components, and the scheduling rules with the highest timing faithfulness. The simulation itself consists in building a chronological track of all the *run-time events* that occurred during the simulated execution. As a result, simulated outputs can be computed as: either user readable in the form of diagrams or reports, or machine readable intended for a subsequent analysis tool. The user interacts with STORM through a user-friendly graphical user interface which is composed of command and display windows.

The notion of time used in STORM is discrete time –i.e., the overall simulation interval is cut into a sequence of indivisible unitary slots such as $[0, 1]$, $[1, 2]$,..., $[t, t + 1]$, etc., and simulation moves forward at each instant 0, 1, ...t, $t + 1$. Thus, at instant t, the next simulation state (for instant $t + 1$) is computed from the current instant.

A.1 Functional Architecture

The internal architecture of the STORM simulator is composed of a set of entities built around a simulation kernel as illustrated in figure A.2. Tasks and data compose the software architecture while processors, memories, and interconnect compose the hardware architecture of STORM simulator. There are as many task, data link and processor entities as specified in the XML input file, and they are automatically instantiated from the library components. Those libraries provide a large collection of task behaviors: recurrent, periodic or aperiodic, with or without activation recording, with or without deadline abort. It's also the case for the data links that enable quite complex synchronization relations between tasks. For the time, two processor types are available: a basic one and another with DPM and DVFS capabilities.

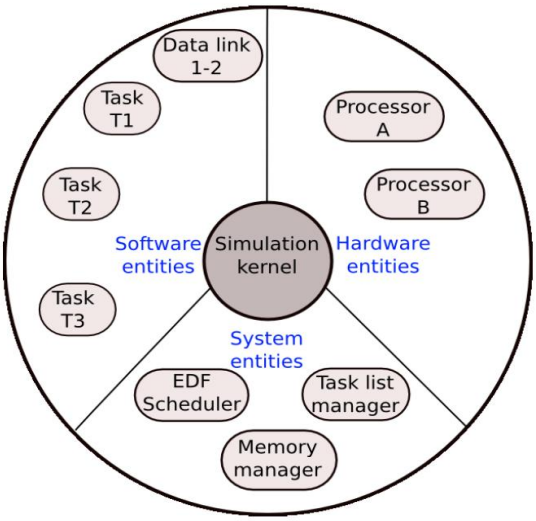


Figure A.2: Functional architecture of STORM simulator.

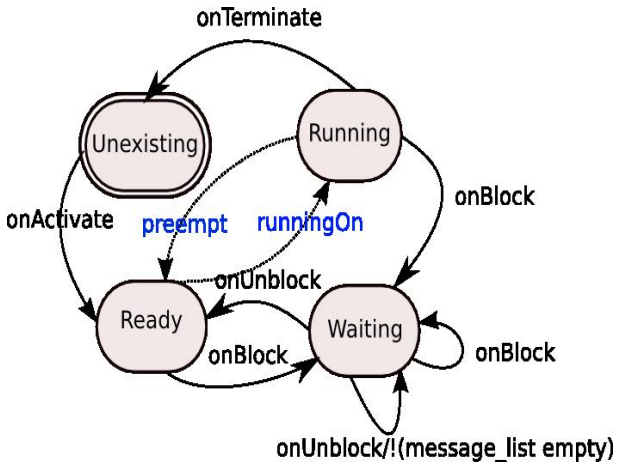


Figure A.3: STORM: various states for application tasks.

A.1.1 Software Entities

Software entities stand for the tasks and data that compose the software architecture for which the simulation is conducted. There are as many task and data link entities as specified in the xml input file. Figure A.4 illustrates and example XML file. It is important to note that these entities capture the behavior of the real components they represent only from a control viewpoint and not a functional one, i.e. no applicative programs run for the tasks, nor true exchanged data values exist. It means that a task is considered only from the execution time point of view and a data from the possible task synchronizations it may give rise to. STORM provides various task types (depending on their activation and deadline control conditions). New task behaviors can be easily specified and added to the tool. Whatever its type, the generic state diagram of a task entity inside the simulator

is shown on figure A.3. At the very beginning, a task is *unexisting*. As soon as its first activation occurs, it becomes ready and falls under the control of a task list manager. Depending on the scheduling decisions, it may *run* and possibly be preempted. On its definitive completion (case of an aperiodic task), the task goes back into the *unexisting* state. On a job completion (for a sporadic or periodic task), it becomes *waiting* until all its execution conditions be met, i.e. only its next release in case of an independent task, but together with the availability of all the data it requires in case of a consumer task. It is the simulator which is in charge of setting the computation time of the jobs each time they are released. By default, the actual execution time (AET) of a new arriving job is simply set equal to the WCET of its task (the value that is specified in the xml input file). But, thanks to the specific attribute calcAET (related to the definition of a task in the xml input file), it is possible to specify another selection strategy (BCET, random value between BCET WCET, value read from an input data log file, etc.). The specification of data in the xml input file of STORM is a way to introduce functional dependence constraints in the form of precedence relations between some producer and consumer tasks. Indeed a task which is the destination of a data has to stay in the waiting state up to (at least) this data becomes available –i.e. in the simplest case, up to the completion of the task which is the source of this data. Possibly a data rate (if specified in the xml input file) needs to be taken into account to determine when the data is really available for consumption.

```
<!-- EDF_P  Earliest Deadline First Preemptive  -->
<!-- Multiprocessor -->
<!-- Input file for DSF tests -->
<SIMULATION duration="1200">

<SCHED className="EDF_P_Scheduler" quantum="1"> </SCHED>
<CPUS>
        <CPU className="storm.Processors.LEATProcessor" name ="CPU A" id="1"></CPU>
        <CPU className="storm.Processors.LEATProcessor" name ="CPU B" id="2"></CPU>
        <CPU className="storm.Processors.LEATProcessor" name ="CPU C" id="3"></CPU>
</CPUS>
<TASKS>
        <TASK className="storm.Tasks.PTask_NAM_A" name ="PTASK A" id="1" period="16" activationDate="0"
        WCET="5" BCET="2" deadline="16"> </TASK>
        <TASK className="storm.Tasks.PTask_NAM_A" name ="PTASK B" id="2" period="20" activationDate="0"
        WCET="5" BCET="2" deadline="20"> </TASK>
        <TASK className="storm.Tasks.PTask_NAM_A" name ="PTASK C" id="3" period="24" activationDate="10"
        WCET="5" BCET="2" deadline="24"> </TASK>
        <TASK className="storm.Tasks.PTask_NAM_A" name ="PTASK D" id="4" period="30" activationDate="10"
        WCET="5" BCET="2" deadline="30"> </TASK>
        <TASK className="storm.Tasks.PTask_NAM_A" name ="PTASK E" id="5" period="40" activationDate="16"
        WCET="12" BCET="10" deadline="40"> </TASK>
        <TASK className="storm.Tasks.PTask_NAM_A" name ="PTASK F" id="6" period="50" activationDate="20"
        WCET="13" BCET="10" deadline="50"> </TASK>
</TASKS>
</DATAS>
        <DATA source="1" destination="2"> rate="1" size="5"></DATA>
</DATAS>
</SIMULATION>
```

Figure A.4: STORM: example XML file.

A.1.2 Hardware Entities

For the moment, hardware architectures are composed of processors only. In the future, it will be most probably extended to take into account other hardware components such as memories and interconnect network, which have some impacts on the behavior and performances of the overall system such as memory banks for instance. Each processor of the considered hardware architecture has its equivalent processor entity. No control is modeled in such an entity but instead it encapsulates some properties stating about its

current activity such as running, idle, or busy. In case of a processor with DVFS (Dynamic Voltage and Frequency Scaling) and DPM(Dynamic Power Management) capabilities, the corresponding entity owns additional properties about its current functionality such as power consumption mode, operating voltage, and frequency etc. together with the functions for updating them.

A.1.3 System Entities

At the moment, system entities are the task list manager, the scheduling algorithms, and the power management policies. The part of the task list manager entity is twofold. It controls the very first release of each task: after a successful comparison between a task activation date and the current time, it produces a specific activation kernel message that is relayed by the kernel towards the concerned task entity and the scheduler entity too. Moreover it manages functional constraints between tasks that appear when data dependencies have been specified: at each simulation slot, depending on the availability or consumption of data, it produces specific task block or unblock kernel messages. The scheduler entity is in charge of sharing the processor(s) between the ready tasks. Its election rules depend on the scheduling strategy it implements. It owns proper queue(s) about ready tasks and manages them thanks to the task state change indications the kernel sends to it and the tick indication too. The election rules it implements may lead it to run and/or to preempt a task. STORM provides most of the standard real-time global event-driven or time-driven schedulers.

A.1.4 Simulation Kernel

The behavior of simulation kernel is a cyclic one –i.e., one cycle for one slot. After a necessary initial step where all the simulation entities are created and the global time variable is initialized, the loop of cycles is entered. Any cycle performs successive steps that come down to manage the watchdogs –i.e., to detect those possible watchdogs that expire and to call the specified function of the specified entity, process all the currently pending kernel messages, call upon the scheduler, manage time passing, ask for the task list manager to operate, and increment the time variable. Furthermore, all along its cycles, the kernel records all the event occurrences and associated data that are relevant for building the simulation outputs.

Appendix B

HyPowMan Scheme: Additional Simulation Results

B.1 Simulation Results Using AsDPM & DSF Experts

In this appendix, we provide additional simulation results on HyPowMan scheme using AsDPM technique (see chapter 4) and DSF technique (see chapter 5) together as experts. These results are obtained for the same target application and simulation settings as used in chapter 6. We observe that the results obtained using DSF and AsDPM as experts are similar to the results obtained in chapter 6 as far as validating the working of HyPowMan is concerned. One exception in these results is that, consistently, AsDPM technique results in better energy savings as compared to DSF technique against all variations –i.e., variation in bcet/wcet ratio, number of tasks, and aggregate utilization. Based on this observation, it is obvious that using only AsDPM technique in this particular case could have yielded best energy savings. Moreover, these results also demonstrate that if the designer/user does not select a single best-performing expert statically based on prior knowledge, HyPowMan scheme can still converge to the single best policy in an online fashion. However, the computational overheads related to HyPowMan scheme could have be avoided in case of statically selected single expert. We have not included these results in chapter 6 as it could have limited the reader from understanding the full potential of HyPowMan scheme. Here, we present these results for reference.

B.1.1 Effect of variations in bcet/wcet ratio

We make the following observations on these results. For bcet/wcet ratio= 1, figure B.1 depicts no change in total energy consumption due to constant dynamic and static power consumption. Since actual execution time remains constant, energy consumed by processors under non-optimized case and under DSF expert alone remains unchanged. AsDPM expert alone, however, saves energy by exploiting the presence of inherent (static) idle intervals. HyPowMan scheme, in this case, converges to AsDPM expert in a straightforward manner as shown in figure B.1. As bcet/wcet ratio decreases (< 1), opportunities for DSF expert to save energy are created as well. Figure B.1 shows that both DSF and AsDPM experts, while working alone, save energy as compared to non-optimized case. For bcet/wcet ratio between 0.5 and 0.9, HyPowMan scheme converges to the best energy savings offered by either expert.

B.1.2 Effect of variations in number of tasks

Similar to the settings in section 6.4.3.2, we double and triple the number of tasks. Results in figure B.2 depict that, increasing the number of tasks increases the energy savings in all cases. These results are very much similar to those obtained in section 6.4.3.2 with the exception that AsDPM expert performed better than DSF expert in all cases.

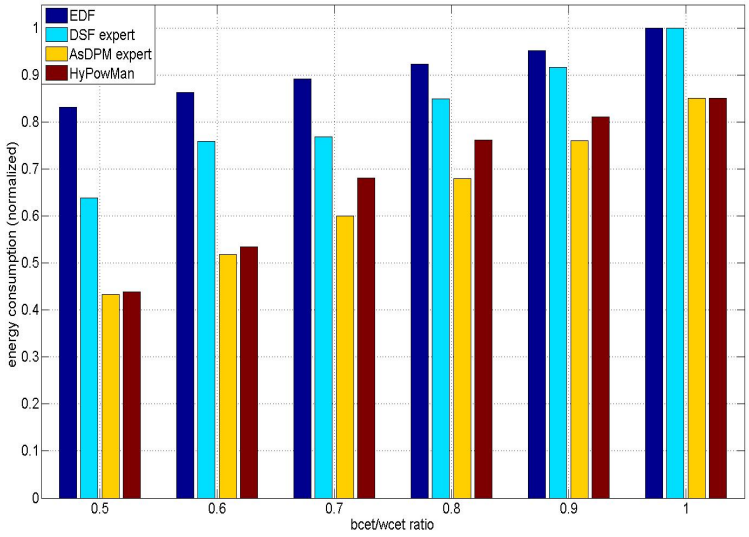


Figure B.1: Simulation results on variation of bcet/wcet ratio.

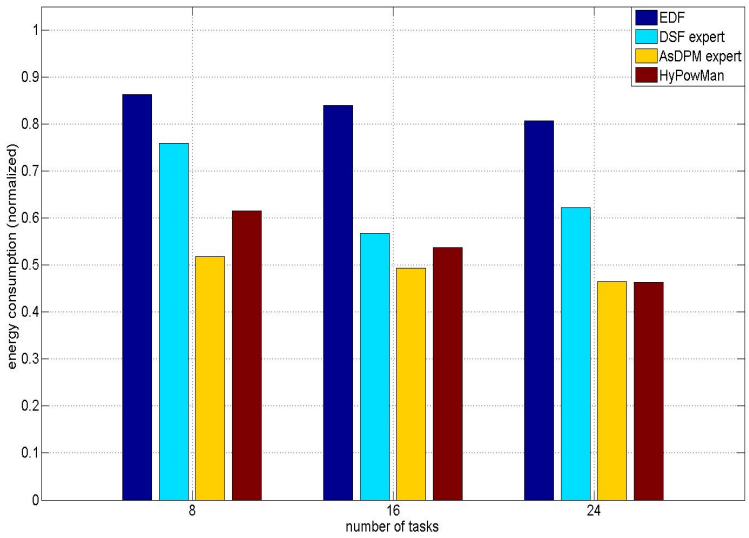


Figure B.2: Simulation results on variation in number of tasks.

B.1.3 Effect of Variations in total utilization

Similar to the settings in section 6.4.3.3, multiple task sets with total utilization varying between 50% (lower workload) and 100% (maximum workload) of platform capacity have been generated in this case. Results in figure B.3 depict that lower aggregate utilization

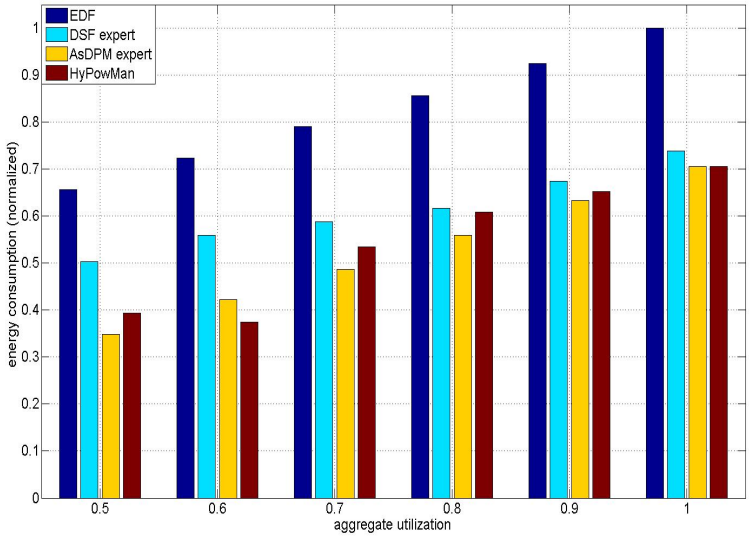


Figure B.3: Simulation results on variation in aggregate utilization.

natually favors energy savings as in case of results presented in section 6.4.3.3 as well. AsDPM expert still performs better than DSF expert in this case.

B.2 Simulation Results Using ccEDF & DSF Experts

In chapter 6, we present our simulation results obtained while using *heterogeneous* techniques –i.e., techniques from different categories such as DPM and DVFS techniques as experts. We have also applied HyPowMan scheme on techniques belonging to the same category, particularly, DVFS techniques. The purpose of these experiments is to demonstrate that HyPowMan scheme is flexible enough to incorporate power management policies from various categories in its expert set. In the following, we present simulation results obtained using *ccEDF* DVFS technique [87] and *DSF* DVFS technique (see chapter 5) as experts on an H.264 video decoder application (slices version) as presented in section 4.5.1. Simulation settings are presented in table B.1.

Table B.1: Simulation settings for variable bcet/wcet ratio

Parameters	Settings
Simulation time	10,000-ms
Number of tasks (n) in task set	7
α for DSF expert	0.70
α for ccEDF expert	0.70
β for DSF expert	0.92
β for ccEDF experts	0.95
bcet/wcet ratio (rule)	bcet+1/3(wcet-bcet)+random[2/3(wcet-bcet)]

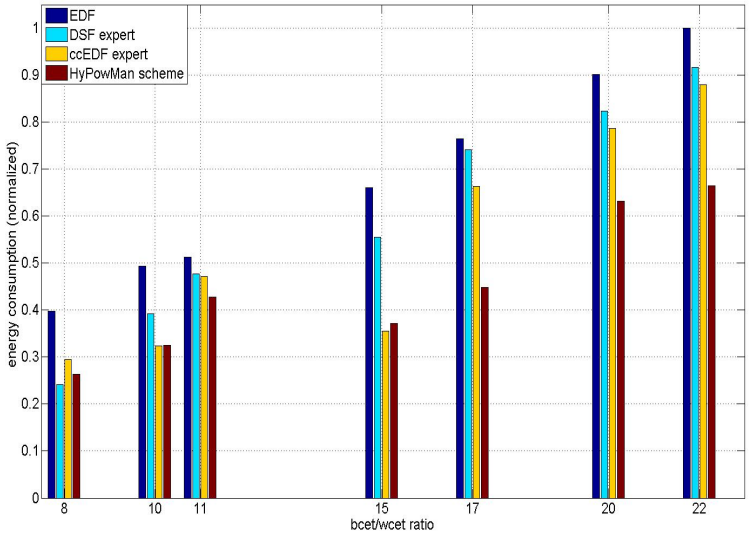


Figure B.4: Simulation results on variation in bcet/wcet ratio.

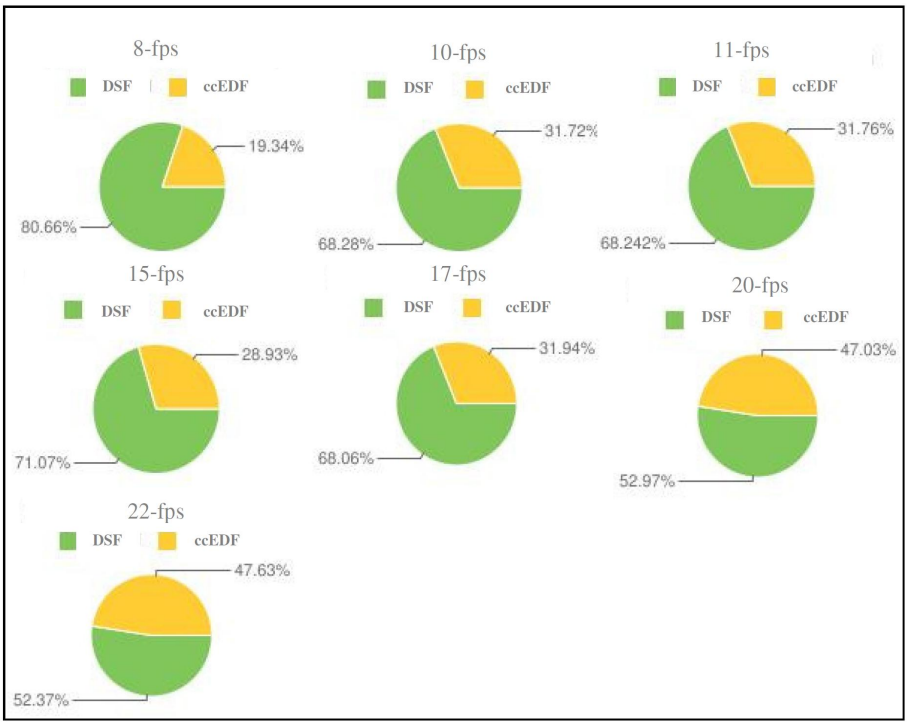


Figure B.5: Simulation results on the usage of experts under the HyPowMan scheme.

Figure B.4 depicts the normalized energy consumption of target application under three cases: non-optimized case –i.e., when no expert is applied, under each expert alone, and under HyPowMan scheme for different QoS (frame rates). Figure B.4 illustrates that both experts, while working alone, save energy with relatively variable gains as compared to non-optimized case. Figure B.4 clearly depicts that HyPowMan scheme converges to the best energy savings offered by either expert. These simulation results demonstrate that HyPowMan scheme can equally work with power management policies from the same as well as different categories and gives energy savings either equivalent to that of best-performing expert in the expert set or even better (which is the case for 11-fps, 17-fps, 20-fps, and 22-fps in figure B.4). We measure best-case energy gains up to 43.77% as compared to non-optimized case and up to 32.45% as compared to best-performing expert.

Figure B.5 depicts the distribution of online usage of experts under HyPowMan scheme. This distribution varies greatly based on instantaneous performance (in response to a power/energy management opportunity) by each expert at any point in time. Higher percentage of usage of any single expert means lesser substitutions are performed online and lesser energy is consumed in substitution. In a converse situation, more energy is consumed and latency to service application tasks is increased. An important observation in figure B.5, with respect to the results presented in figure B.4, is that for higher frame rates, ccEDF expert also becomes effective and consequently being used more frequently by HyPowMan. This indicates that variations in workload can shift the balance of performance amongst different DVFS policies at runtime.

Bibliography

[1] Tarek A. AlEnawy and Hakan Aydin. Energy-aware task allocation for rate monotonic scheduling. In *procs. of IEEE Symposium, RTAS-2005*, pages 213–223, 2005. 115

[2] AMD. AMD Technical Docs, 2010. http://support.amd.com. 85

[3] James H. Anderson and Sanjoy K. Baruah. Energy-aware implementation of hard-real-time systems upon multiprocessor platforms. In *In Proceedings of the ISCA 16th International Conference on Parallel and Distributed Computing Systems*, pages 430–435, 2002. 12, 17

[4] James H. Anderson and Sanjoy K. Baruah. Energy-efficient synthesis of periodic task systems upon identical multiprocessor platforms. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, ICDCS '04, pages 428–435, Washington, DC, USA, 2004. IEEE Computer Society. 17

[5] James H. Anderson, Vasile Bud, and UmaMaheswari C. Devi. An edf-based scheduling algorithm for multiprocessor soft real-time systems. In *Proceedings of the 17th Euromicro Conference on Real-Time Systems*, pages 199–208, Washington, DC, USA, 2005. IEEE Computer Society. 35

[6] James H. Anderson and Anand Srinivasan. Mixed pfair/erfair scheduling of asynchronous periodic tasks. In *Proceedings of the 13th Euromicro Conference on Real-Time Systems*, ECRTS '01, pages 76–, Washington, DC, USA, 2001. IEEE Computer Society. 54

[7] Bjorn Andersson, Sanjoy Baruah, and Jan Jonsson. Static-priority scheduling on multiprocessors. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, RTSS '01, pages 93–, Washington, DC, USA, 2001. IEEE Computer Society. 22, 35, 39

[8] Bjorn Andersson and Eduardo Tovar. Multiprocessor scheduling with few preemptions. In *Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, RTCSA '06, pages 322–334, Washington, DC, USA, 2006. IEEE Computer Society. 20, 22, 32, 33, 34

[9] ARM. ARM Architecture, 2010. http://www.arm.com/. 85, 130, 150

[10] Hakan Aydin, Rami Melhem, Daniel Mossé, and Pedro Mejía-Alvarez. Power-aware scheduling for periodic real-time tasks. *IEEE Trans. Comput.*, 53:584–600, May 2004. 8, 28, 77, 86, 87, 89, 98, 99, 105, 116, 128, 139, 146, 149

[11] Hakan AYDIN, Rami MELHEM, Daniel MOSSÉ, and P. MEJIA-ALVAREZ. Power-aware scheduling for periodic real-time tasks. In *IEEE Transactions on Computers*, pages 584 – 600. IEEE, vol. 53, n°5, 2004. 51

[12] Hakan Aydin and Qi Yang. Energy-aware partitioning for multiprocessor real-time systems. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, IPDPS '03, pages 113.2–, Washington, DC, USA, 2003. IEEE Computer Society. 89

[13] Sanjoy Baruah, Neil Cohen, Greg Plaxton, and Don Varvel. Proportionate progress: A notion of fairness in resource allocation. In *Algorithmica*, page 600 625, 1996. 4, 31, 35, 129, 136, 149

[14] Sanjoy K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: a notion of fairness in resource allocation. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, STOC '93, pages 345–354, New York, NY, USA, 1993. ACM. 52

[15] Sanjoy K. Baruah, Aloysius K. Mok, and Louis E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *In Proceedings of the 11th Real-Time Systems Symposium*, pages 182–190. IEEE Computer Society Press, 1990. 44

[16] Luca Benini, Alessandro Bogliolo, and Giovanni De Micheli. A survey of design techniques for system-level dynamic power management. In Giovanni De Micheli, Rolf Ernst, and Wayne Wolf, editors, *Readings in hardware/software co-design*, pages 231–248. Kluwer Academic Publishers, Norwell, MA, USA, 2002. 26, 58, 60

[17] Luca Benini and Giovanni de Micheli. *Dynamic Power Management: Design Techniques and CAD Tools*. Springer, 1997. 26

[18] Luca Benini and Giovanni De Micheli. *Dynamic Power Management: Design Techniques and CAD Tools*. Springer; 1st edition (November 30, 1997), 1997. 60

[19] Muhammad Khurram Bhatti, Cécile Belleudy, and Michel Auguin. An inter-task real time dvfs scheme for multiprocessor embedded systems. In *Proceedings of IEEE international conference on Design and Architectures for Signal and Image Processing, DASIP*, 2010. 115

[20] Muhammad Khurram Bhatti, Cécile Belleudy, and Michel Auguin. Power management in real time embedded systems through online and adaptive interplay of dpm and dvfs policies. In *proceedings of International Conference on Embedded and Ubiquitous Computing, EUC'*10, EUC'10, 2010. 8, 140, 146

[21] Enrico Bini, Giorgio C. Buttazzo, and Giuseppe M. Buttazzo. Rate monotonic analysis: The hyperbolic bound. *IEEE Trans. Comput.*, 52:933–942, July 2003. 20

[22] Enrico Bini, Giorgio C. Buttazzo, and Giuseppe Lipari. Speed modulation in energy-aware real-time systems. In *IEEE Proceedings of the Euromicro Conference on Real-Time Systems, ECRTS*, 2005. 99

[23] Alan Burns and Andy Wellings. *Real-Time Systems and Programming Languages (Third Edition). Ada 95, Real-Time Java and Real-Time POSIX*. Hardback - 611 pages. Addison Wesley Longmain, 2001. 19, 22

[24] John M. Calandrino, James H. Anderson, and Dan P. Baumberger. A hybrid real-time scheduling approach for large-scale multicore platforms. In *Proceedings of the 19th Euromicro Conference on Real-Time Systems, ECRTS'07*, pages 247–256, 2007. 32, 34, 35, 124, 144

[25] Jian-Jia Chen and Chin-Fu Kuo. Energy-efficient scheduling for real-time systems on dynamic voltage scaling (dvs) platforms. In *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, RTCSA '07, pages 28–38, Washington, DC, USA, 2007. IEEE Computer Society. 89

[26] Hui Cheng and Steve Goddard. Online energy-aware i/o device scheduling for hard real-time systems. In *Proceedings of the conference on Design, automation and test in Europe: Proceedings*, DATE '06, pages 1055–1060, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association. 60

[27] Hui Cheng and Steve Goddard. Sys-edf: a system-wide energy-efficient scheduling algorithm for hard real-time systems. In *International Journal of Embedded Systems*, pages 141 – 151. Inderscience, Volume 4, Number 2 / 2009, 2009. 60, 79, 82, 109

[28] Hyeonjoong Cho, Binoy Ravindran, and E. Douglas Jensen. An optimal real-time scheduling algorithm for multiprocessors. In *Proceedings of the 27th IEEE International Real-Time Systems Symposium*, pages 101–110, Washington, DC, USA, 2006. IEEE Computer Society. 4, 31, 52, 129, 136, 149

[29] Eui-Young Chung, Luca Benini, Alessandro Bogiolo, and Giovanni De Micheli. Dynamic power management for non-stationary service requests. In *Proceedings of the conference on Design, automation and test in Europe*, DATE '99, New York, NY, USA, 1999. ACM. 27, 60

[30] Ayse K. Coskun, David Atienza, Tajana Simunic Rosing, Thomas Brunschwiler, and Bruno Michel. Energy-efficient variable-flow liquid cooling in 3d stacked architectures. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '10, pages 111–116, 3001 Leuven, Belgium, Belgium, 2010. European Design and Automation Association. 5, 130, 136, 150

[31] Victor De La Luz, Mahmut Kandemir, and Ugur Sezer. Improving off-chip memory energy behavior in a multi-processor, multi-bank environment. In *Proceedings of the 14th international conference on Languages and compilers for parallel computing*, LCPC'01, pages 100–114, Berlin, Heidelberg, 2003. Springer-Verlag. 80

[32] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M. J. Irwin. Dram energy management using sof ware and hardware directed power mode control. In *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, HPCA '01, pages 159–, Washington, DC, USA, 2001. IEEE Computer Society. 80

[33] M. L. Dertouzos and A. K. Mok. Multiprocessor online scheduling of hard-real-time tasks. *IEEE Trans. Softw. Eng.*, 15:1497–1506, December 1989. 19

[34] Michael Dertouzos and Aloysius Mok. Multiprocessor scheduling in a hard real-time environment. In *IEEE Transactions on Software Engineering*, page 1497 1506, 1989. 22, 35

[35] Vinay Devadas and Hakan Aydin. On the interplay of dynamic voltage scaling and dynamic power management in real-time embedded applications. In *Proceedings of the 8th ACM international conference on Embedded software*, EMSOFT '08, pages 99–108, New York, NY, USA, 2008. ACM. 27, 59, 60, 109

[36] Vinay Devadas and Hakan Aydin. Real-time dynamic power management through device forbidden regions. In *Proceedings of the 2008 IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 34–44, Washington, DC, USA, 2008. IEEE Computer Society. 59, 60

[37] Gaurav Dhiman and Tajana Simunic Rosing. Dynamic power management using machine learning. In *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, ICCAD '06, pages 747–754, New York, NY, USA, 2006. ACM. 8, 108, 109, 110, 140, 146

[38] Francois Dorin, Patrick Meumeu Yomsi, Joel Goossens, and Pascal Richard. Semi-partitioned hard real-time scheduling with restricted migrations upon identical multiprocessor platforms. In *Operating Systems (cs.OS), arXiv:*1006.2637*v*1 *[cs.OS], Cornell University Library Archives, Technical Report*, June, 2010. 34, 35, 36, 37

[39] R. ERNST and W. YE. Embedded program timing analysis based on path clustering and architecture classification. In *the proceedings of International Conference on Computer-Aided Design (ICCAD 97)*, pages 598 – 604, 1997. 50, 90

[40] Rolf Ernst and Weisong Ye. Embedded program timing analysis based on path clustering and architecture classification. In *Proceedings of the 1997 IEEE/ACM international conference on Computer-aided design*, ICCAD '97, pages 598–604, Washington, DC, USA, 1997. IEEE Computer Society. 28, 86

[41] Amir H. Farrahi, Gustavo E. Téllez, and Majid Sarrafzadeh. Memory segmentation to exploit sleep mode operation. In *Proceedings of the 32nd annual ACM/IEEE Design Automation Conference*, DAC '95, pages 36–41, New York, NY, USA, 1995. ACM. 80

[42] Nathan Wayne Fisher. The Multiprocessor Real-Time Scheduling of General Task Systems, 2007. PhD Thesis, University of North-Carolina at Chapel Hill,. 3, 11, 18, 20, 135

[43] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory*, pages 23–37, London, UK, 1995. Springer-Verlag. 110

[44] Kenji Funaoka, Shinpei Kato, and Nobuyuki Yamasaki. A context cache replacement algorithm for pfair scheduling. In *Proceedings of the 15th International Conference on Real-Time and Network Systems (RTNS)*, pages 57 – 64, 2007. 46

[45] Kenji Funaoka, Shinpei Kato, and Nobuyuki Yamasaki. Dynamic voltage and frequency scaling for optimal real-time scheduling on multiprocessors. In *Proceedings of Industrial Embedded Systems, 2008. SIES 2008. International Symposium*, pages 27–33, Le Grande Motte, France, 2008. 89

[46] Kenji Funaoka, Shinpei Kato, and Nobuyuki Yamasaki. Energy-efficient optimal real-time scheduling on multiprocessors. In *Proceedings of the 2008 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing*, pages 23–30, Washington, DC, USA, 2008. IEEE Computer Society. 89, 90, 108

[47] Shelby Hyatt Funk. Edf scheduling on heterogeneous multiprocessors. In *PhD thesis, department of computer science*. University of North Carolina at Chapel Hill, 2004. 3, 11, 18, 20, 22, 135

[48] Bruno Gaujal and Nicolas Navet. Dynamic voltage scaling under edf revisited. *Real-Time Syst.*, 37:77 – 97, October 2007. 28, 85, 125, 145

[49] S. Gochman, R. Ronen, I. Anati, A. Berkovis, T. Kurts, A. Naveh, A. Saeed, Z. Sperber, and R. C. Valentine. The intel pentium m processor: Microarchitecture and performance. In *Proceedings of Intel Technology Journal, Vol. 7, Issue 2*, pages 21–36, 2003. 89

[50] Flavius Gruian. System-level design methods for low-energy architectures containing variable voltage processors. In *Proceedings of the First International Workshop on Power-Aware Computer Systems-Revised Papers*, PACS '00, pages 1–12, London, UK, 2001. Springer-Verlag. 89

[51] Flavius GRUIAN. Energy-centric scheduling for real-time systems. In *PhD thesis*. http://www.cs.lth.se/home/Flavius Gruian, 2002. 86

[52] Kwang Soo Hong and Joseph Y.-T. Leung. On-line scheduling of real-time tasks. In *IEEE Real-Time Systems Symposium*, page 244 250, Huntsville,Alabama, 1988. 22, 35

[53] Chi-Hong Hwang and Allen C.-H. Wu. A predictive system shutdown method for energy saving of event-driven computation. *ACM Trans. Des. Autom. Electron. Syst.*, 5:226–241, April 2000. 58

[54] Intel. Intel Technical Docs, 2010. http://www.intel.com. 85

[55] S. Irani, R. Gupta, and S. Shukla. Competitive analysis of dynamic power management strategies for systems with multiple power savings states. In *Proceedings of the conference on Design, automation and test in Europe*, DATE '02, pages 117–, Washington, DC, USA, 2002. IEEE Computer Society. 27, 60

[56] Sandy Irani and Kirk R. Pruhs. Algorithmic problems in power management. *SIGACT News*, 36:63–76, June 2005. 4, 136

[57] Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Online strategies for dynamic power management in systems with multiple power-saving states. *ACM Trans. Embed. Comput. Syst.*, 2:325–346, August 2003. 26, 57, 60

[58] Ravindra Jejurikar and Rajesh Gupta. Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems. In *Proceedings of the 2004 international symposium on Low power electronics and design*, ISLPED '04, pages 78–81, New York, NY, USA, 2004. ACM. 79, 109

[59] Ravindra Jejurikar, Cristiano Pereira, and Rajesh Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. In *Proceedings of the 41st annual Design Automation Conference*, DAC '04, pages 275–280, New York, NY, USA, 2004. ACM. 90

[60] David Johnson. Near-optimal bin packing algorithms. In *PhD thesis*, Department of Mathematics, Massachusetts Institute of Technology, 1973. 4, 6, 32, 124, 136, 138, 144

[61] Anna R. Karlin, Mark S. Manasse, Lyle A. McGeoch, and Susan Owicki. Competitive randomized algorithms for non-uniform problems. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, SODA '90, pages 301–309, Philadelphia, PA, USA, 1990. Society for Industrial and Applied Mathematics. 115

[62] Shinpei Kato and Nobuyuki Yamasaki. Portioned edf-based scheduling on multiprocessors. In *Proceedings of the 8th ACM international conference on Embedded software*, EMSOFT '08, pages 139–148, New York, NY, USA, 2008. ACM. 32, 33, 34, 35, 36, 37, 124, 144

[63] Shinpei Kato, Nobuyuki Yamasaki, and Yutaka Ishikawa. Semi-partitioned scheduling of sporadic task systems on multiprocessors. In *Proceedings of the 2009 21st Euromicro Conference on Real-Time Systems*, pages 249–258, Washington, DC, USA, 2009. IEEE Computer Society. 20, 32, 33, 34, 44, 124, 144

[64] Minyoung Kim and Soonhoi Ha. Hybrid run-time power management technique for real-time embedded system with voltage scalable processor. In *Proceedings of the 2001 ACM SIGPLAN workshop on Optimization of middleware and distributed systems*, OM '01, pages 11–19, New York, NY, USA, 2001. ACM. 109

[65] C. M. Krishna and Yann-Hang Lee. Voltage-clock-scaling adaptive scheduling techniques for low power in hard real-time systems. *IEEE Trans. Comput.*, 52:1586–1593, December 2003. 89

[66] Woo-Cheol Kwon and Taewhan Kim. Optimal voltage allocation techniques for dynamically variable voltage processors. In *Proceedings of the 40th annual Design Automation Conference*, DAC '03, pages 125–130, New York, NY, USA, 2003. ACM. 89

[67] Alvin R. Lebeck, Xiaobo Fan, Heng Zeng, and Carla Ellis. Power aware page allocation. *SIGARCH Comput. Archit. News*, 28:105–116, November 2000. 80

[68] Seongsoo Lee and Takayasu Sakurai. Run-time voltage hopping for low-power real-time systems. In *Proceedings of the 37th Annual Design Automation Conference*, DAC '00, pages 806–809, New York, NY, USA, 2000. ACM. 28

[69] Benjamin Lesage, Damien Hardy, and Isabelle Puaut. Shared data cache conflicts reduction for wcet computation in multi-core architectures. In *Proceedings of 18th International Conference on Real-Time and Network Systems*, Toulouse, France, 2010. 14

[70] Benjamin Lesage and Isabelle Puaut. Estimation of cache related migration delays for multi-core processors with shared instruction caches. In *Proceedings of 17th International Conference on Real-Time and Network Systems*, pages 45 – 54, Paris, France, 2009. 14

[71] Chung L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of ACM*, 20:46–61, January 1973. 15, 19, 20, 22, 129, 149

[72] Marvell. Marvell's XScale Microarchitecture. http://www.marvell.com/. 29, 85

[73] Andreas Merkel and Frank Bellosa. Balancing power consumption in multiprocessor systems. *SIGOPS Oper. Syst. Rev.*, 40:403–414, April 2006. 82, 84

[74] Aloysius K. Mok. Task management techniques for enforcing ed scheduling on a periodic task set. *Proceedings of 5th IEEE Workshop on Real-Time Software and Operating Systems*, pages 42–46, 1988. 22

[75] Daniel Mosse, Hakan Aydin, Bruce Childers, and Rami Melhem. Compiler-assisted dynamic power-aware scheduling for real-time applications. In *In Workshop on Compilers and Operating Systems for Low Power*, 2000. 89

[76] Daniel MOSSE, Hakan AYDIN, Buce CHILDERS, and Rami MELHEM. Compiler-assisted dynamic power-aware scheduling for real-time applications. In *Workshop on Compiler and Operating Sys for Low-Power*, 2000. 86

[77] Farooq Muhammad. Ordonnancement de tâches efficace et â complexitè maîtrisèe pour des systèmes temps rèel. In *PhD thesis*. University of Nice-Sophia Antipolis, 2009. 3, 4, 11, 31, 37, 44, 52, 54, 135, 136

[78] Fabrizio Mulas, David Atienza, Andrea Acquaviva, Salvatore Carta, Luca Benini, and Giovanni De Micheli. Thermal balancing policy for multiprocessor stream computing platforms. *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, 28:1870–1882, December 2009. 79, 82, 130

[79] N. NAVET and B. GAUJAL. *Ordonnancement temps reel et minimisation de la consommation de energie*. Chapter-4 in System Temps Reel -Volume 2. Kluwer Publishers, 2006. 26, 86, 87

[80] Vincent Nelis. Energy-Aware Real-Time Scheduling in Embedded Multiprocessor Systems, 2010. PhD Thesis, Université Libre de Bruxelles, Belgium. 11, 25

[81] Ozcan Ozturk and Mahmut Kandemir. Nonuniform banking for reducing memory energy consumption. In *Proceedings of the conference on Design, Automation and Test in Europe - Volume 2*, DATE '05, pages 814–819, Washington, DC, USA, 2005. IEEE Computer Society. 80

[82] Ozcan Ozturk, Mahmut Kandemir, and I. Koleu. Reducing memory energy consumption of embedded applications that process dynamically allocated data. In *IEEE transactions on computer-aided design of integrated circuits and systems, vol. 25, no.9*, 2006. 80

[83] G. A. Paleologo, L. Benini, A. Bogliolo, and G. De Micheli. Policy optimization for dynamic power management. In *Proceedings of the 35th annual Design Automation Conference*, DAC '98, pages 182–187, New York, NY, USA, 1998. ACM. 26, 60

[84] G. A. Paleologo, L. Benini, A. Bogliolo, and G. De Micheli. Policy optimization for dynamic power management. In *Proceedings of the 35th annual Design Automation Conference*, DAC '98, pages 182–187, New York, NY, USA, 1998. ACM. 60

[85] Minkyu Park, Sangchul Han, Heeheon Kim, Seongje Cho, and Yookun Cho. Comparison of deadline-based scheduling algorithms for periodic real-time tasks on multiprocessor*this work is supported in part by brain korea 21 project and in part by ict. *IEICE - Trans. Inf. Syst.*, E88-D:658–661, March 2005. 19

[86] PHERMA. ANR project Pherma (Reference: ANR-06-ARFU06-003) , 2007 - 2010. http://pherma.irccyn.ec-nantes.fr. 28, 70, 128, 148, 155

[87] Padmanabhan Pillai and Kang G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, SOSP '01, pages 89–102, New York, NY, USA, 2001. ACM. 28, 89, 102, 115, 163

[88] Pherma project technical report. Validation of the offline and online optimization tools, 2010. http://pherma.irccyn.ec-nantes.fr/publications.php. 29, 69, 102, 103, 105

[89] QEMU. QEMU, 2010. http://wiki.qemu.org/. 105, 130, 150

[90] Qinru Qiu and Massoud Pedram. Dynamic power management based on continuous-time markov decision processes. In *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, DAC '99, pages 555–561, New York, NY, USA, 1999. ACM. 60

[91] Gang Quan and Xiaobo Sharon Hu. Energy efficient dvs schedule for fixed-priority real-time systems. *ACM Trans. Embed. Comput. Syst.*, 6, September 2007. 89

[92] Dinesh Ramanathan, Sandy Irani, and Rajesh Gupta. Latency effects of system level power management algorithms. In *Proceedings of the 2000 IEEE/ACM international conference on Computer-aided design*, ICCAD '00, pages 350–356, Piscataway, NJ, USA, 2000. IEEE Press. 26, 27

[93] Rambus. Rambus Inc.,, 1999. http://www.rambus.com/. 80

[94] Saowanee Saewong and Ragunathan Rajrajkumar. Optimal static voltage-scaling for real-time systems, 2007. 89

[95] Jaewon Seo, Taewhan Kim, and Ki-Seok Chung. Profile-based optimal intra-task voltage scheduling for hard real-time applications. In *Proceedings of the 41st annual Design Automation Conference*, DAC '04, pages 87–92, New York, NY, USA, 2004. ACM. 86

[96] Dongkun Shin and Jihong Kim. Optimizing intra-task voltage scheduling using data flow analysis. In *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, ASP-DAC '05, pages 703–708, New York, NY, USA, 2005. ACM. 86

[97] Youngsoo Shin and Kiyoung Choi. Power conscious fixed priority scheduling for hard real-time systems. In *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, DAC '99, pages 134–139, New York, NY, USA, 1999. ACM. 89, 98

[98] Youngsoo Shin, Kiyoung Choi, and Takayasu Sakurai. Power optimization of real-time embedded systems on variable speed processors. In *Proceedings of the 2000 IEEE/ACM international conference on Computer-aided design*, ICCAD '00, pages 365–368, Piscataway, NJ, USA, 2000. IEEE Press. 86, 89

[99] Patrick M. Shriver, Maya B. Gokhale, Scott D. Briles, Dong-In Kang, Michael Cai, Kevin McCabe, Stephen P. Crago, and Jinwoo Suh. *A power-aware, satellite-based parallel signal processing scheme*, pages 243–259. Kluwer Academic Publishers, Norwell, MA, USA, 2002. 89

[100] Sandeep K. Shukla and Rajesh K. Gupta. A model checking approach to evaluating system level dynamic power management policies for embedded systems. In *Proceedings of the Sixth IEEE International High-Level Design Validation and Test Workshop (HLDVT'01)*, HLDVT '01, pages 53–, Washington, DC, USA, 2001. IEEE Computer Society. 27, 60

[101] Tajana Simunic, Luca Benini, Andrea Acquaviva, Peter Glynn, and Giovanni De Micheli. Dynamic voltage scaling and power management for portable systems. In *Proceedings of the 38th annual Design Automation Conference*, DAC '01, pages 524–529, New York, NY, USA, 2001. ACM. 109

[102] Pushkar Singh and Vinay Chinta. Survey report on dynamic power management. In *Survey report of the University of Illinois, Chicago (ECE Department)*, Chicago, USA, 2008. 26, 58, 59

[103] Kevin Skadron, Mircea R. Stan, Karthik Sankaranarayanan, Wei Huang, Sivakumar Velusamy, and David Tarjan. Temperature-aware microarchitecture: Modeling and implementation. *ACM Trans. Archit. Code Optim.*, 1:94–125, March 2004. 4, 136

[104] Arvind Sridhar, Alessandro Vincenzi, Martino Ruggiero, Thomas Brunschwiler, and David Atienza Alonso. 3d-ICE: Fast compact transient thermal modeling for 3D-ICs with inter-tier liquid cooling. In *Proceedings of the 2010 (ICCAD 2010)*, volume 1, pages 463–470, New York, 2010. ACM and IEEE Press. 5, 130, 136, 150

[105] Anand Srinivasan. Efficient and flexible fair scheduling of real-time tasks on multiprocessors. In *PhD thesis, department of computer science*. University of North Carolina at Chapel Hill, 2003. 3, 4, 11, 135, 136

[106] Anand Srinivasan and James Anderson. Fair scheduling of dynamic task systems on multiprocessors. *J. Syst. Softw.*, 77:67 – 80, July 2005. 4, 35, 136

[107] Mani B. Srivastava, Anantha P. Chandrakasan, and R. W. Brodersen. Predictive system shutdown and other architectural techniques for energy efficient programmable computation. *IEEE Trans. Very Large Scale Integr. Syst.*, 4:42–55, March 1996. 58

[108] STORM. STORM simulation tool. http://storm.rts-software.org. 8, 28, 47, 69, 99, 129, 140, 150, 155

[109] Vishnu Swaminathan and Krishnendu Chakrabarty. Energy-conscious, deterministic i/o device scheduling in hard . . . *IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS*, 22(7):858, 2003. 59

[110] Vishnu Swaminathan and Krishnendu Chakrabarty. Pruning-based, energy-optimal, deterministic i/o device scheduling for hard real-time systems. *ACM Trans. Embed. Comput. Syst.*, 4:141–167, February 2005. 59, 61

[111] Vishnu Swaminathan and Krishnendu Chakrabarty. Pruning-based, energy-optimal, deterministic i/o device scheduling for hard real-time systems. *ACM Trans. Embed. Comput. Syst.*, 4:141–167, February 2005.

[112] Vishnu Swaminathan, Krishnendu Chakrabarty, and S. S. Iyengar. Dynamic i/o power management for hard real-time systems, 2001. 59

[113] Vishnu Swaminathan, Krishnendu Chakrabarty, and S. S. Iyengar. Dynamic i/o power management for hard real-time systems. In *Proceedings of the ninth international symposium on Hardware/software codesign*, CODES '01, pages 237–242, New York, NY, USA, 2001. ACM. 60

[114] Pengliu Tan, Jian Shu, and Zhenhua Wu. A hybrid real-time scheduling approach on multi-core architectures. In *Journal of software, vol. 5, No. 9, September 2010*, pages 958–965. Academy Publisher, 2010. 34

[115] Thales. Thales group (France), 2010. http://www.thalesgroup.com/. 70, 72, 104, 126

[116] Real time systems group. IRCyN research laboratory, University of Nantes, France., 2009. 28, 155

[117] Transmeta. Transmera. http://www.transmeta.com/. 85

[118] Richard Urunuela, Anne-Marie Déplanche, and Yvon Trinquet. Simulation for multiprocessor real-time scheduling evaluation. In *Proceedings of 7th EUROSIM Congress on Modelling and Simulation, Eurosim'10*, 2010. 155

[119] Richard Urunuela, Anne-Marie Déplanche, and Yvon Trinquet. Storm: A simulation tool for real-time multiprocessor scheduling evaluation. In *Proceedings of 15th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA'10*, 2010. 155

[120] Nicolas Ventroux and Raphaël David. Scmp architecture: an asymmetric multiprocessor system-on-chip for dynamic applications. In *Proceedings of the Second International Forum on Next-Generation Multicore/Manycore Technologies*, IFMT '10, pages 6:1–6:12, New York, NY, USA, 2010. ACM. 128, 148

[121] Weixun Wang and Prabhat Mishra. Predvs: preemptive dynamic voltage scaling for real-time systems using approximation scheme. In *Proceedings of the 47th Design Automation Conference*, DAC '10, pages 705–710, New York, NY, USA, 2010. ACM. 86

[122] Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. Scheduling for reduced cpu energy. In *Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*, OSDI '94, Berkeley, CA, USA, 1994. USENIX Association. 89

[123] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David B. Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter P. Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem - overview of methods and survey of tools. In *ACM Transaction on Embedded Computing Systems, vl. 7, No.03*, 2008. 14

[124] Peng Yang, Chun Wong, Paul Marchal, Francky Catthoor, Dirk Desmet, Diederik Verkest, and Rudy Lauwereins. Energy-aware runtime scheduling for embedded-multiprocessor socs. volume 18, pages 46–58, Los Alamitos, CA, USA, September 2001. IEEE Computer Society Press. 26, 89

[125] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, FOCS '95, pages 374–, Washington, DC, USA, 1995. IEEE Computer Society. 89

[126] Baoxian Zhao and Hakan Aydin. Minimizing expected energy consumption through optimal integration of dvs and dpm. In *Proceedings of the 2009 International Conference on Computer-Aided Design*, ICCAD '09, pages 449–456, New York, NY, USA, 2009. ACM. 108

[127] Xiliang Zhong and Cheng-Zhong Xu. System-wide energy minimization for real-time tasks: lower bound and approximation. In *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, ICCAD '06, pages 516–521, New York, NY, USA, 2006. ACM. 89

[128] Dakai Zhu, Nevine AbouGhazaleh, Daniel Mosse, and Rami Melhem. Power aware scheduling for and/or graphs in multi-processor real-time systems. In *In Proc. of The Intl Conference on Parallel Processing*, pages 593–601, 2002. 89

[129] Dakai Zhu, Rami Melhem, and Bruce R. Childers. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems. *IEEE Trans. Parallel Distrib. Syst.*, 14:686–700, July 2003. 89

[130] Jianli Zhuo and Chaitali Chakrabarti. System-level energy-efficient dynamic task scheduling. In *Proceedings of the 42nd annual Design Automation Conference*, DAC '05, pages 628–631, New York, NY, USA, 2005. ACM. 79

[131] Jianli Zhuo, Chaitali Chakrabarti, and Naehyuck Chang. Energy management of dvs-dpm enabled embedded systems powered by fuel cell-battery hybrid source. In *Proceedings of the 2007 international symposium on Low power electronics and design*, ISLPED '07, pages 322–327, New York, NY, USA, 2007. ACM. 109

Abstract

Modern real-time applications have become more sophisticated and complex in their behavior over the time. Contemporaneously, multiprocessor architectures have emerged. Multiprocessor systems, due to their autonomy and reliability, face critical problem of energy consumption. To address this issue in real-time systems, many software-based approaches have emerged. This thesis proposes new techniques for energy-efficient scheduling of multiprocessor systems. Our first contribution is a hierarchical scheduling algorithm that allows restricted migration of tasks. This algorithm aims at reducing the sub-optimality of global EDF algorithm. The second contribution of this thesis is a dynamic power management technique called Assertive Dynamic Power Management (AsDPM). This technique is an admission control technique for real-time tasks, which decides when exactly a ready task shall execute, thereby reducing the number of active processors. The third contribution of this dissertation is a DVFS technique, referred as Deterministic Stretch-to-Fit (DSF) technique, which falls in the category of inter-task DVFS techniques. Both DPM and DVFS techniques are efficient for specific operating conditions. However, they often outperform each other when these conditions change. Our fourth and final contribution is a generic power/energy management scheme, called Hybrid Power Management (HyPowMan) scheme. This scheme, instead of designing new power/energy management techniques for specific operating conditions, takes a set of well-known existing policies. At runtime, the best-performing policy for given workload is adapted by HyPowMan scheme through machine-learning approach.

Résumé

Les applications temps réel modernes deviennent plus exigeantes en termes de ressources et de débit amenant la conception d'architectures multiprocesseurs. Ces systèmes, des équipements embarqués au calculateur haute performance, sont, pour des raisons d'autonomie et de fiabilité, confrontés des problèmes cruciaux de consommation d'énergie. Pour ces raisons, cette thèse propose de nouvelles techniques d'optimisation de la consommation d'énergie dans l'ordonnancement de systèmes multiprocesseur. La premiére contribution est un algorithme d'ordonnancement hiérarchique á deux niveaux qui autorise la migration restreinte des tâches. Cet algorithme vise á réduire la sous-optimalité de l'algorithme global EDF. La deuxiéme contribution de cette thèse est une technique de gestion dynamique de la consommation nommée Assertive Dynamic Power Management (AsDPM). Cette technique, qui régit le contrôle d'admission des tâches, vise á exploiter de manière optimale les modes repos des processeurs dans le but de réduire le nombre de processeurs actifs. La troisiéme contribution propose une nouvelle technique, nommée Deterministic Stretch-to-Fit (DSF), permettant d'exploiter le DVFS des processeurs. Les gains énergétiques observés s'approchent des solutions déjà existantes tout en offrant une complexité plus réduite. Ces techniques ont une efficacité variable selon les applications, amenant á définir une approche plus générique de gestion de la consommation appelée Hybrid Power Management (HyPowMan). Cette approche sélectionne, en cours d'exécution, la technique qui répond le mieux aux exigences énergie/performance.